HAL
open science

Détection temps réel de postures humaines par fusion d'images 3D

Wassim Filali

To cite this version:

Wassim Filali. Détection temps réel de postures humaines par fusion d'images 3D. Automatique / Robotique. Universite Toulouse III Paul Sabatier, 2014. Français. NNT : . tel-01137267

HAL Id: tel-01137267

https://hal.science/tel-01137267

Submitted on 30 Mar 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*

Discipline ou spécialité : *Systèmes embarqués et robotique*

Présentée et soutenue par *Wassim FILALI*

Le *07 Novembre 2014*

Titre : *Détection temps réel de postures humaines par fusion d'images 3D.*

Unité de recherche : *LAAS-CNRS (Groupe RAP, Groupe N2IS)*

École doctorale : *EDSYS*

Directeur(s) de Thèse : *Frédéric LERASLE, Jean-Louis BOIZARD*

Rapporteurs : *Paul CHECCHIN, Alberto IZAGUIRRE*

Examinateurs : *Mohamed AKIL, Michel DEVY*

Proverbe

Rien ne sert de courir.

Remerciements

Je remercie mes parents, ma famille, mes amis.

Je remercie mes encadrants sans qui ce travail n'aurait pas été possible.

Je remercie Michel DEVY pour tout ce qu'il a fait pour moi.

Un grand merci à l'équipe RAP et à tous les membres du LAAS avec qui j'ai passé une période inoubliable.

Résumé

Nous présentons dans ce manuscrit le contenu de la thèse, centrée sur un projet de recherche en vision. Nous présentons l'étude de l'état de l'art de la reconstruction de posture et leurs contextes d'application associés. Nous partons des systèmes embarqués et caméras intelligentes et nous focalisons sur la vision par ordinateur et son utilisation pour la reconnaissance d'activités humaines. Ensuite, nous nous intéressons à la reconstruction de posture, car cela représente l'élément-clé du processus de reconnaissance. Ces travaux de thèse se sont appuyés sur les dernières avancées technologiques typiquement l'avènement du capteur RGB-D type Kinect. Nous avons développé un algorithme de fusion bas niveau de multiples capteurs de profondeur.

Les challenges sous-jacents sont liés à plusieurs phénomènes : (1) aux occultations dues à l'utilisation d'un capteur unique, ensuite, (2) à la complexité combinatoire en fonction du nombre de postures à apprendre, et (3) aux contraintes imposées à l'intégration du système. Nous avons abordé chacun de ces points.

L'utilisation de plusieurs capteurs permet par évidence de s'affranchir de la majorité des problèmes d'auto-occultation et offre de meilleures garanties de robustesse et complétude sur la scène observée. Nous avons donc élaboré une technique de fusion bas niveau qui représente a priori la contribution principale de la thèse. Nous avons adapté une technique d'apprentissage fondée sur des forêts de décision. Notre algorithme s'appuie sur notre propre base d'apprentissage élaborée à partir de notre plate-forme multi-kinect couplée à un système commercial de capture de mouvement.

Les deux principales spécificités sont la fusion de données sensorielles et l'apprentissage supervisé. Plus précisément, la fusion des données est décrite par les étapes de capture, segmentation et voxellisation qui génèrent une reconstruction 3D de l'espace occupé. L'apprentissage s'appuie sur le formalisme des forêts de décision en utilisant un descripteur approprié. Des expérimentations et le réglage (« tuning ») des paramètres de l'apprentissage ont également été réalisés.

Une comparaison avec l'état de l'art a été menée de façon qualitative et quantitative avec des résultats concluants au niveau de la précision des articulations reconstruites.

L'étude algorithmique a été approfondie sur un environnement PC et a permis de cibler une sous-partie des modalités à intégrer dans notre système. L'intégration matérielle a consisté en une étude et comparaison des différentes approches disponibles. Les FPGA sont une plate-forme permettant de répondre aux critères de performance et d' « embarquabilité », car ils délivrent une puissance permettant de réduire le coût CPU. Ceci nous a permis d'apporter une contribution qui consiste à hiérarchiser la conception et développer une couche de « modules » intermédiaires. Une comparaison a été menée entre une fonctionnalité de détection d'arrière-plan, intégrée sur PC, GPU et FPGA puis son implémentation sur FPGA a été détaillée.

Le document se termine par la conclusion et les perspectives qui font l'objet de futures investigations et prospectives professionnelles.

Abstract

This thesis manuscript presents a reflection of the different research investigations, rooted in computer vision, which were carried out during the life time of this PhD research. It presents a comprehensive study of the state-of-the-art in human posture reconstruction, its contexts, and associated applications. The underlying research focuses on utilization of computer vision techniques for human activity recognition based on embedded system technologies and intelligent camera systems. It also focuses on human posture reconstruction as it plays a key role in subsequent activity recognition. In this work, we have relied on the latest technological advances in sensor technology, specifically on the advent of Kinect, an RGB-D sensor from Microsoft, to realize a low-level sensor fusion algorithm to fuse the outputs of multiple depth sensors for human posture reconstruction.

In this endeavor, the different challenges encountered are: (1) occlusions when using a single sensor; (2) the combinatorial complexity of learning a high dimensional space corresponding to human postures; and finally, (3) embedded systems constraints. The proposed system addresses and consequently resolves each of these challenges.

The fusion of multiple depth sensors gives better result than individual sensors as the fusion alleviates the majority of occlusions by resolving many incoherencies thus by guaranteeing improved robustness and completeness on the observed scene. In this manuscript, we have elaborated the low-level fusion strategy which makes up the main contribution of this thesis. We have adopted a learning technique based on decision forests. Our algorithm is applied on our own learning dataset acquired with our multi-platform kinect coupled to a commercial motion capture system.

The two main principal features are sensor data fusion and supervised learning. Specifically, the data fusion technique is described by acquisition, segmentation, and voxelization which generates a 3D reconstruction of the occupied space. The supervised learning is based on decision forests and uses appropriate descriptors extracted from the reconstructed data. Various experiments including specific parameter learning (tuning) runs have been realized.

Qualitative and quantitative comparative human articulation reconstruction precision evaluations against the state-of-the-art strategies have also been carried out.

The different algorithms have been implemented on a personal computer environment which helped to analyze the essential parts that needs hardware embedded integration. The hardware integration consisted of studying and comparing multiple approaches. FPGA is a platform that meets both the performance and embeddability criteria as it provides resources that reduce CPU cost. This allowed us to make a contribution which constitutes a hierarchically prioritized design via a layer of intermediary modules. Comparative studies have also been done using background subtraction implementation as a benchmark integrated on PC, GPU, and FPGA (the FPGA implementation has been presented in detail).

Finally, the manuscript terminates with concluding remarks and future research project and investigation prospects.

Publications

Filali, W., Masse, J., Lerasle, F., Boizard, J. L., & Devy, M.
Human motion capture using 3D reconstruction based on multiple depth data.
In *IEEE International Conference on Systems, Man, and Cybernetics* (SMC 2013). Manchester (Angleterre).

Chaabani, H., Filali, W., Simon, T., & Lerasle, F.
Body Pixel Classification by Neural Network.
In *International Conference on Intelligent Robotics and Applications* (ICIRA 2012). Montreal (Canada).

Filali, W., Botero, D., Devy, M., & Boizard, J. L.
SOPC components for real time Image processing : Rectification, Distortion correction and Homography.
In *IEEE Workshop on Electronics, Control, Measurement and Signals* (ECMS 2011). Liberec (République Tchèque).

Botero, D., Devy, M., Boizard, J. L., & Filali, W.
Real-time architecture on FPGA for obstacle detection using inverse perspective mapping.
In *IEEE International Conference on Electronics, Circuits, and Systems* (ICECS 2011). Beyrouth (Liban).

Manzano Ibarra, M., Filali, W., Devy, M., Botero, D., Fourniols, J., & Boizard, J. L.
Obstacle avoidance by a multi-cameras system.
In *International Workshop on Electronics, Control, Modelling, Measurement and Signals* (ECMS 2009). Mondragon (Espagne).

Devy, M., Manzano, M. I., Boizard, J. L., Lacroix, P., Filali, W., & Fourniols, J. Y.
Integrated subsystem for Obstacle detection from a belt of micro-cameras.
In *IEEE International Conference on Advanced Robotics* (ICAR 2009). Munich (Allemagne).

Table des matières

Remerciements 5
Résumé 6
Abstract 7
Publications 8
Table des matières 9
Chapitre 1 : Introduction, contexte et motivations 14
1.1 Introduction 14
1.2 Le projet région CameraNet 15
1.3 Cahier des charges 15
1.4 État de l'art et positionnement de nos travaux 16
1.4.1 Le capteur de profondeur kinect 16
1.4.2 Le marché du kinect 17
1.5 Notre problématique 18
1.6 Résumé de notre approche 19
1.7 Plan du manuscrit 20
Chapitre 2 : État de l'art et considérations expérimentales 22
2.1 État de l'art sur la capture de mouvement humain sans marqueurs 22
2.1.1 Reconstruction de posture par vision monoculaire passive 22
2.1.2 Estimation de posture par vision multi oculaire passive 23
2.1.3 Estimation de posture par mono capteur de profondeur et labellisation des parties corporelles 25
2.1.4 Travaux de Shotton et al 26
2.1.5 Estimation de posture par multi-capteurs de profondeur 30
2.1.6 Récapitulatif et résumé de notre approche 30
2.2 Description de notre plate-forme multi capteurs 32
2.2.1 Réseau de capteurs 32
2.2.2 Notre environnement d'expérimentation 32
2.3 Acquisition des données 34
2.3.1 Acquisition 34
2.3.2 Position des marqueurs pour la MOCAP 34
2.3.3 Codage et stockage des données 35

2.4 Étalonnage géométrique et synchronisation temporelle 35
2.4.1 Étalonnage du système de MOCAP 35
2.4.2 Étalonnage des capteurs de profondeur 36
2.4.3 Étalonnage des capteurs d'images couleur 36
2.4.4 Étalonnage géométrique du système multi capteurs RGB-D versus MOCAP 36
2.4.5 Synchronisation temporelle : système MOCAP versus capteurs RGB-D 37
2.5 Bases de données acquises 38
2.6 Conclusion 40
Chapitre 3 : Formalisation et implémentation de notre algorithme 42
3.1 Introduction 42
3.2 Synopsis de notre approche 42
3.2.1 Synopsis 42
3.2.2 Notre approche vs. Shotton *et al.* 43
3.2.3 Étapes de traitement 44
3.3 Formalisation de notre algorithme 46
3.3.1 Voxellisation 46
3.3.2 Caractérisation des descripteurs de taille fixe 48
3.3.3 Classification à descripteur de taille fixe 49
3.3.4 Formalisation des « random forests » 51
3.3.5 Récapitulatif des paramètres libres et variables 55
3.3.6 Estimation de la position des centres des parties corporelles 56
3.4 Implémentation 57
3.4.1 Paramètres libres et voxellisation 57
3.4.2 Paramètres libres de la labellisation des voxels 58
3.5 Caractérisation des paramètres libres 59
3.5.2 Nombre de Candidats par nœud 60
3.5.3 Le sous-échantillonnage de la base des voxels à apprendre 61
3.5.4 Paramètres intrinsèques à l'arbre de décision 63
3.5.5 Paramètre de seuil du gain d'information 64
3.5.6 Fenêtre des vecteurs de descripteurs 65
3.5.7 Nombre d'arbres dans la forêt 66
3.5.8 Vote vs. pondération 68
3.5.9 Récapitulatif des paramètres libres 70
3.6 Conclusion 70

Chapitre 4 Évaluations et étude comparative 72
4.1 Évaluations 72
4.1.1 Base de données et paramètres libres associés 72
4.1.2 Observations sur le processus d'apprentissage 74
4.1.3 Évaluations qualitatives 76
4.1.4 Évaluations quantitatives sur la classification 78
4.1.5 Évaluations quantitatives sur les postures reconstruites 81
4.2 Notre approche vs. OpenNI 82
4.2.1 Paramètres d'apprentissage 82
4.2.2 Observations sur la reconstruction par OpenNI 83
4.2.3 Comparaison qualitative 84
4.2.4 Comparaison quantitative 87
4.3 Conclusion 89
Chapitre 5 : Intégration matérielle sur FPGA 90
5.1 État de l'art sur la méthodologie et l'architecture 90
5.1.1 État de l'art sur la méthodologie pour les Systèmes embarqués 91
5.1.2 Répartition des architectures des calculateurs 93
5.1.3 Analyse des différentes architectures 95
5.1.4 Exemple de matériel dédié à la détection de posture. 98
5.2 Modélisation du système avec SysML 100
5.3 Environnement du système embarqué reprogrammable 102
5.3.1 Introduction 102
5.3.2 Environnement de développement 102
5.3.3 Composants de base 105
5.3.4 Paramètres influençant la performance 108
5.3.5 Exemples de pipelines et leurs performances 109
5.4 Placement des ressources avec les solutions retenues 111
5.4.1 Architecture multi xtion sur PC 112
5.4.2 Présentation de l'architecture GPU 113
5.4.3 Architecture smart caméras multi vues sur FPGA 115
5.4.4 Comparaison d'une fonction sur CPU, FPGA, GPU 115
5.5 Détails de l'implémentation de la fonction soustraction de fond sur FPGA 118
5.5.1 Formalisation de la soustraction de fond 118

5.5.2 Représentation de la soustraction de fond par blocs matériels 118
5.5.3 Implémentation sous QuartusII - QSys 120
5.5.4 Chronogrammes de fonctionnement 122
5.5.5 Évaluation..... 123
5.6 Conclusion..... 123
Chapitre 6 : Conclusion et perspectives..... 126
Table des figures 129
Liste des tableaux..... 133
Glossaire..... 135
Bibliographie 137

Chapitre 1 : Introduction, contexte et motivations

Ce chapitre porte sur l'intégration des systèmes embarqués et la détection optique de posture. Nous y présentons notre projet et nous nous positionnons vis-à-vis de l'état de l'art. Nous dégageons les problématiques à traiter et nous concluons par la présentation du plan du manuscrit.

1.1 Introduction

La loi de Moore (Moore 1965), correspondant à l'augmentation des performances des calculateurs, a permis de franchir un nouveau palier i.e. les systèmes embarqués et autonomes se dotent d'assez de puissance leur permettant d'exécuter des algorithmes complexes, auparavant réservés aux machines fixes. Ceci ouvre la porte à de nouvelles applications.

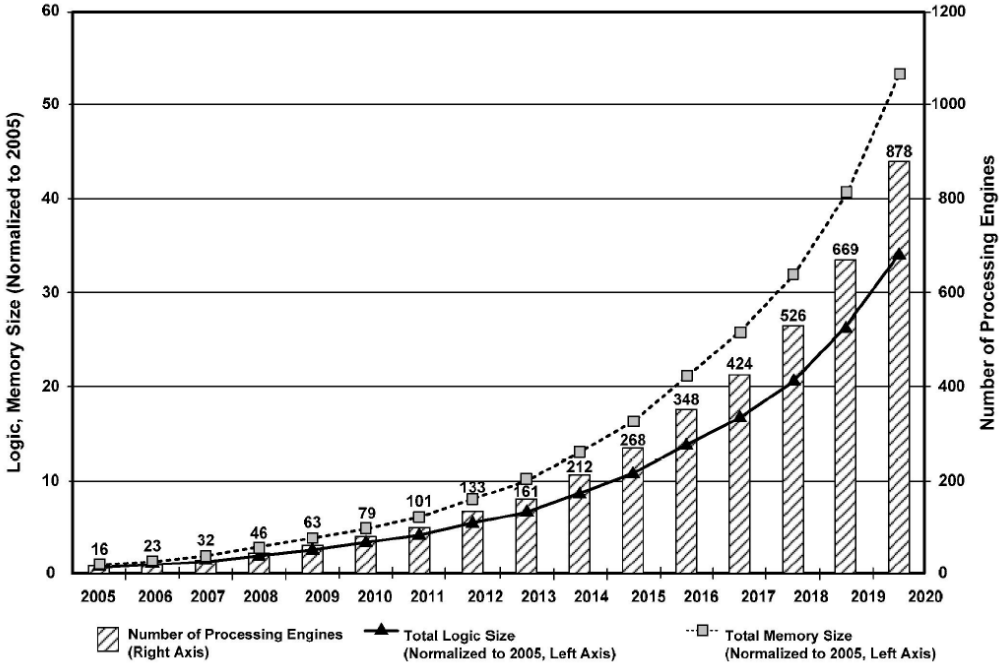

Figure 1-1 : L'édition 2005 de l'ITRS (International Technology Roadmap for Semiconductors) du nombre de processeurs contenus dans un système embarqué.

La figure 1-1 est accompagnée d'une analyse optimiste de (Leibson 2006) qui suppose qu'une fois la fréquence des générations de processeurs a cessé de croître, c'est la conception des systèmes qui prend la relève avec un nombre croissant de processeurs multicores. D'autres émettent un avis plus pessimiste sur la fin de la montée en échelle des processeurs multicores

(Esmaeilzadeh et al. 2011) pour qui la montée en échelle est limitée, car il est impossible d'accélérer une application unique au-delà d'un certain seuil quelle que soit l'architecture ou le nombre de processeurs.

La Vision par Ordinateur est une application qui peut profiter d'une large parallélisation, ce qui la rend apte à profiter de l'évolution du matériel en processeurs multicores. Par ailleurs, la Vision par Ordinateur représente un élément-clef dans l'évolution de l'interaction des systèmes robotiques et des systèmes embarqués intelligents.

La thèse s'inscrit dans cette problématique. Il s'agit de développer un système de vision multi-capteurs du point de vue algorithmique avec une perspective d'intégration matérielle. La finalité est de reconstruire la posture de la personne en temps réel. Ce genre d'application est répandu *e.g.* dans les jeux vidéo, la vidéosurveillance et la garde à domicile de personnes âgées. Cette dernière application vise la détection automatique de situations dangereuses sans intrusion dans la vie privée de la personne âgée.

D'abord, nous présentons le projet adossé à ces travaux dans la première partie de ce chapitre 1 avec son cahier des charges. Ensuite, nous abordons l'état de l'art. Nous présentons le kinect dont l'introduction sur le marché grand public a impacté ce domaine de recherche. Nous finirons ce chapitre 1 par le plan du manuscrit.

1.2 Le projet région CameraNet

Ces travaux s'inscrivent dans un projet de réseau de caméras intelligentes dénommé « CameraNet ». Il a été financé conjointement par la région et par le CNRS. C'est un projet de recherche et de transfert de technologie du Conseil Régional Midi-Pyrénées. L'avènement du kinect et les travaux algorithmiques de Microsoft sur la reconstruction de posture ont largement inspiré ce projet.

Le cadre applicatif est orienté vers l'aide au maintien à domicile de personnes âgées en utilisant des capteurs optiques d'ambiance. Les verrous sont à la fois algorithmique et matériel, car la finalité est d'instrumenter un espace privatif avec des capteurs autonomes. Le projet CameraNet était initialement motivé par ces deux verrous.

1.3 Cahier des charges

L'objectif est donc de reconstruire la posture d'une personne pour détecter les situations de danger. Le système doit éviter l'intrusion dans la vie privée de la personne et ne doit donc pas consister à visualiser la scène par une tierce personne. La scène doit être traitée automatiquement et en temps réel pour éviter la nécessité d'enregistrement des séquences ainsi que pour donner une alerte de façon instantanée. Les fausses alertes doivent être minimisées ce qui milite pour une reconstruction de postures robuste. Nous privilégions alors un réseau de capteurs RGB-D. La plate-forme multi-capteurs est supposée calibrée et une fusion bas niveau des données sensorielles délivrées est privilégiée.

La reconstruction de postures est essentielle à notre système préalablement à la détection de chutes ou plus largement postures dangereuses , il est important de pouvoir caractériser la dynamique des parties corporelles, pour cela, une évaluation de la posture plusieurs fois par seconde est nécessaire.

Une étude préliminaire d'une solution matérielle est à effectuer avec adéquation associée entre algorithme et architecture. Cette adéquation doit permettre la satisfaction de contraintes temps réel ainsi que la minimisation de la consommation énergétique pour un système censé tourner en permanence. Le système doit pouvoir viser la minimisation de l'encombrement pour la solution à long terme. Ceci n'est pas applicable aux prototypes ou aux cartes de développement, mais aux systèmes qui en découleront. Une solution orientée vers un matériel spécifique doit donc être envisagée.

1.4 État de l'art et positionnement de nos travaux

Notre état de l'art s'appuie initialement sur quelques considérations relatives aux systèmes embarqués et techniques de Vision par Ordinateur puis se focalise sur notre problématique de reconstruction de postures humaines.

1.4.1 Le capteur de profondeur kinect

La figure 1-2 représente les deux capteurs de profondeur grand public (kinect à gauche et Asus à droite). Les composants constituant le capteur de profondeur sont fournis par le même constructeur Primesense. La différence réside dans les accessoires (le kinect est motorisé) et dans les logiciels qui les pilotent.

Figure 1-2 : Capteurs de profondeur grand public.

En stéréovision (vision passive), il est classique d'utiliser une paire de caméras calibrées pour obtenir une image de disparité qui nous renseigne sur la profondeur.

Cette technique est peu privilégiée dans les applications commerciales à bas coût, c'est à cause de plusieurs inconvénients :

- La stéréovision est insensible aux zones non texturées.
- Elle requiert souvent l'ajout d'un éclairage pour s'affranchir des conditions ambiantes d'illumination et la présence de texture dans la scène.
- Le temps de calcul des techniques de corrélation dense est supérieur à celui des techniques éparses.

Ces capteurs de profondeur (vision active) que l'on présente ici pallient ces inconvénients grâce à des approches simples mais judicieuses.

D'abord, comme le capteur requiert de la lumière, il est équipé d'un projecteur qui éclaire la scène. Cela permet de percevoir des objets, même dans le noir. Afin que cette lumière ne soit pas gênante, elle a été définie dans une zone infrarouge (830 nm : *infrarouge* dans la sous division *proche infrarouge*). La lumière envoyée est structurée, ce qui permet de percevoir les

[1] http://www.microsoft.com/en-us/kinectforwindows/

[2] http://www.asus.com/Multimedia/Motion_Sensor/Xtion_PRO_LIVE/

objets non texturés. Cette lumière structurée se présente sous forme d'un ensemble de rayons calibrés tels que illustré dans la figure 1-3.

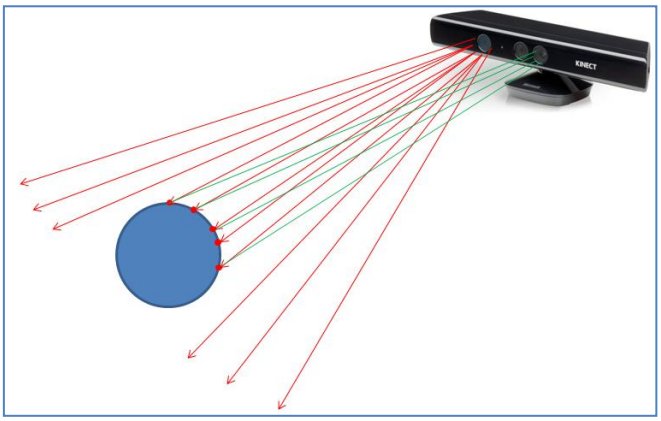

Figure 1-3 : Principe de fonctionnement du Kinect.

Ces rayons calibrés évitent le besoin d'une seconde caméra. Ceci signifie que les équations des rayons projetés sont caractérisées lors de l'étalonnage à l'usine. Ceci consiste concrètement à calculer les équations des rayons dans le repère de la caméra. La disparité n'est donc pas calculée à partir de deux caméras ayant deux rayons perçus du même point, mais à partir d'un rayon projeté et de son image perçue sur le plan image et donc la droite de vue associée.

Le coût CPU propre à la stéréovision a été pallié grâce à un investissement dans une solution complètement matérielle avec un ASIC[3].

1.4.2 Le marché du kinect

Le kinect a été introduit sur le marché en Novembre 2010. 8 millions d'unités ont été vendues lors des deux premiers mois et 18 millions en janvier 2012, 14 mois après le lancement. (Cruz et al. 2012) résument très bien l'historique entre la date de commercialisation, et le dévouement de la communauté de développeurs « open source » pour rendre cet équipement utilisable par tous. À la base, il n'était supporté que par la console de jeux Xbox 360. Une multitude de librairies et d'applications ont progressivement été développées telles que libfreenect, OpenNI MS Kinect SDK, PCL parmi d'autres. Le tableau 1-1 présenté par (Cruz et al. 2012) liste les fonctionnalités de ces librairies.

[3] Application Specific Integrated Circuit

	OpenNI	libfreenect	MS Kinect SDK	PCL
Skeleton	X		X	
Hand location	X			
Face tracking			X	
Body gesture recognition	X		X	
Hand gesture recognition	X			
Scene analyser	X		X	
Speach recognition			X	
3D reconstruction				X
Filtering				X
Multi-device	X			X
Camera calibration	X		X	
Motor control		X		
Multi-platform	X	X		X

Tableau 1-1 : Exemple de librairies associées aux capteurs de profondeur et leurs fonctions (Cruz et al. 2012).

Un état de l'art sur la capture de mouvement sans marqueurs est détaillé dans le chapitre 2. Nous focalisons ici sur deux approches caractéristiques privilégiant caméras passives ou actives. Le tableau 1-2 compare les travaux de (Hofmann & Gavrila 2009) sur un système de caméras classiques et (Shotton et al., 2011b) sur un capteur kinect. L'utilisation de plusieurs points de vue avec des caméras classiques (non actives) par Hofmann et al ainsi que le suivi par logique différée en font des travaux de premier plan.

Caméras classiques [Hofmann et al. 2009]	**Kinect [Shotton et al., 2011]**
Haut du corps (tout le corps prévu en ***future works***)	Tout le corps
15-20s (temps réel prévu en ***future works***)	Super temps réel : 10 % de ressources pour 30 FPS
1 personne (haut du corps) (plusieurs personnes prévues en ***future works***)	2 personnes (4 segmentées sans squelette)

Tableau 1-2 : Comparaison du Kinect avec Hofmann et al 2009.

Les travaux de (Shotton et al., 2011b) ont été mis en valeur par une application commerciale à bas coût destinée au grand public. Ce qui signifie qu'elle est dans une phase de maturité technique avancée. Dans ce même axe, on peut citer la collaboration entre Plagemann et Ganapathi dans (Plagemann & Koller 2010; Ganapathi et al. 2010). Ils reconstituent également la posture en détectant les parties du corps et en utilisant une image de profondeur issue d'une caméra ToF (*Time of Flight*) capteur très onéreux avec une plus faible résolution.

1.5 Notre problématique

Notre approche tire parti de l'avènement du Kinect et s'inspire des travaux associés de Shotton et al. Citons en l'occurrence :

- L'***utilisation de la profondeur*** générée par le kinect fait abstraction de la variabilité des textures des différents vêtements et est insensible à la couleur de peau, ce qui posait problème dans les algorithmes fondés sur la vision uniquement (segmentation d'arrière-plan, détection de couleur de peau, …). La profondeur a déjà été utilisée avec des systèmes de stéréovision passive, mais la résolution a été limitée comparée au système de vision active de la kinect.

- La ***détection des parties du corps*** réduit la complexité du problème. L'apprentissage visuel 2D a pour complexité l'ordre de dimension des degrés de liberté du corps en plus de la variation de l'apparence. L'apprentissage des parties du corps avec la profondeur est réduit à la localité de la partie en question. Cela réduit sa complexité. En effet, on associe les données aux parties corporelles. Le traitement réparti se trouve simplifié. Il y a par ailleurs l'utilisation de techniques d'apprentissage qui seront évoquées dans la suite de ce manuscrit.

Nos travaux sont par ailleurs motivés par les constats suivants :

- Les approches mono capteurs sont sujettes à occultation, si la personne est dos au capteur par exemple. Un autre inconvénient est que le champ de vue d'un capteur unique est restreint.
- Pour notre problématique, il n'existe pas de base publique multi-capteurs de profondeur avec vérité-terrain associée. On s'appuie ici classiquement sur un système commercial d'instrumentation du sujet humain (appelé ici MOCAP) pour la mesure de données de captures de mouvement. Notre protocole expérimental est détaillé dans la partie qui suit.
- Si le calcul de disparité pour générer la profondeur en temps réel à la fréquence du flux vidéo est exécuté par logiciel, il demanderait des ressources qui dépassent les capacités de machines standard. Le kinect calcule l'appariement avec du matériel dédié, mais la classification des parties du corps et la détection de posture sont exécutées en logiciel et demandent des ressources acceptables mais quand même importantes (dual core 2.66 GHz avec 2 GO de RAM).

1.6 Résumé de notre approche

Pour lever les verrous cités précédemment, nous allons d'abord remédier aux inconvénients dus à la mono vision. Notre première contribution consiste à capturer une ***base de données réelle disposant de multiples flux vidéos issus de capteurs RGB-D et de données MOCAP*** exploitées durant l'apprentissage et les évaluations. Cette base contient des données de positions 3D enregistrées avec un système de MOCAP qui nous donne une position précise de la vérité-terrain des parties du corps et des centres des articulations. On effectue un étalonnage géométrique des multiples capteurs en accord avec le système de MOCAP ainsi qu'une synchronisation temporelle entre les divers flux.

La complexité combinatoire dans l'association des données/parties corporelles est résolue par un apprentissage par partie corporelle et une stratégie de voxellisation de la silhouette 3D par fusion ***bas niveau de divers flux kinect***. Cet espace 3D est alors segmenté en parties corporelles grâce à la vérité-terrain délivrée par le MOCAP. ***Un apprentissage de descripteurs de voxels via un arbre de décision*** est alors mis en œuvre pour classer les voxels en parties corporelles ; la reconstruction de posture repose alors sur cette labellisation des voxels de la silhouette et un algorithme de « mean shift » pour localiser les articulations. Des évaluations quantitatives ont été réalisées sur notre approche ; elles s'appuient ici encore sur la vérité terrain délivrée par le MOCAP.

Enfin, au-delà de l'algorithmique, des investigations visant à intégrer de multiples fonctionnalités de ***vision*** sur ***FPGA ont été réalisées***. La finalité est un système embarqué assurant les étapes de classification des parties corporelles puis de reconstruction 3D des articulations. Il libère ainsi les systèmes en aval de surcharges de calcul.

On peut donc résumer la liste de nos contributions par :

- La mise en place d'une plate-forme multi kinect et l'acquisition d'une base de données avec vérité-terrain.
- La reconstruction de posture multi kinect par voxellisation préalable de la silhouette.
- L'étude de l'adéquation entre cet algorithme et l'architecture dédiée avec résultats préliminaires.

1.7 Plan du manuscrit

Le manuscrit est structuré comme suit. Le chapitre 2 décrit notre protocole expérimental et les bases de flux vidéo multi-kinects enregistrés. L'état de l'art sur la capture de mouvement sans marqueurs i.e. par techniques de Vision par Ordinateur est alors présenté. Notre approche tire parti des travaux de (Shotton et al., 2011b) qui sont alors décrits ici. Les atouts d'une stratégie multi-kinects sont alors listés. Le chapitre présente alors notre plate-forme pour l'acquisition de base de données multi-kinects conjointement à une référence MOCAP. Cette base de données relatives à des mouvements humains très diversifiés, ne présente pas, à notre connaissance, d'équivalent dans la littérature.

Les chapitres 3 et 4 détaillent le volet algorithmique de nos travaux ; on détaille notre algorithme de reconstruction de posture par fusion bas niveau avec multi capteurs de profondeur et les évaluations associées.

Le chapitre 3 décrit donc la formalisation puis l'implémentation. Nos divers choix sont ici justifiés tandis que le formalisme des « random forests » est ici appréhendé. Le descriptif de notre implémentation permet d'exhiber les paramètres libres de notre approche.

Le chapitre 4 détaille l'influence et le réglage (« tuning ») des paramètres libres et montre des évaluations quantitatives et qualitatives. Notre approche est alors logiquement comparée de façon qualitative/quantitative avec l'approche mono kinect de (Shotton et al., 2011b).

Le chapitre 5 reprend le synopsis algorithmique pour motiver les travaux sur le volet matériel et donc l'intégration dans un système embarqué. On présente d'abord un état de l'art sur les architectures et aussi la méthodologie sous-jacente. Les différentes architectures sont alors catégorisées afin de justifier nos choix technologiques. Ensuite, on présente une solution de référence, suivie de la modélisation de notre solution avec SysML. Notre étude permet d'adopter une approche de rencontre au point milieu (« *meet in the middle* »). Notre algorithme de reconstruction est mis en regard des solutions matérielles existantes et des contraintes applicatives inhérentes. D'abord, on constitue l'environnement de développement avec des fonctionnalités unitaires. Ensuite, la méthode de rencontre au point milieu permet d'associer par raffinements successifs, des solutions matérielles aux fonctions de traitement qui ont été identifiées.

Le dernier chapitre présente les conclusions et perspectives associées à ces travaux.

Chapitre 2 : État de l'art et considérations expérimentales

Dans ce chapitre, on commence par la présentation d'un état de l'art plus approfondi sur la capture de mouvement sans marqueurs. On en dégage les limitations des approches mono capteur et on justifie notre approche multi capteurs. Ensuite, on présente la mise en œuvre permettant l'acquisition et la construction des bases de données avec vérité-terrain. On termine par le descriptif de notre base de données.

2.1 État de l'art sur la capture de mouvement humain sans marqueurs

La capture de mouvement sans marqueurs est une technique qui utilise la vision par ordinateur. Elle peut varier selon les applications qui en ont besoin, ainsi que selon les contraintes associées. À titre d'exemple, l'analyse des séquences vidéo a posteriori (film, TV, surveillance), opère dans un contexte mono vision. Quand il est possible d'instrumenter l'environnement, l'utilisation de plusieurs caméras est privilégiée. Cela permet de lever des ambiguïtés dues à l'occultation. L'utilisation de la profondeur est une technique qui diffère de la vision passive. Elle fait appel à des ressources matérielles différentes. Elle est employée dans des applications ludiques comme dans un cadre de jeux vidéo par exemple. Dans ce contexte, par contre, il est difficile d'instrumenter l'environnement avec plus d'un capteur. Notre contexte applicatif est la supervision des postures pour la surveillance des personnes âgées. On peut y justifier l'utilisation des caméras de profondeur ainsi que leur multiplicité. Il reste donc à lever les verrous algorithmiques et les verrous d'intégration matérielle.

Pour profiter de ce qui existe déjà dans la littérature, on va présenter dans les lignes qui suivent différentes approches pour noter les points qui nous intéressent ainsi que les limitations observées.

2.1.1 Reconstruction de posture par vision monoculaire passive

La détermination de la posture d'une personne en 3D en utilisant des images monoculaires représente un réel défi. Même avec un a priori de la forme du modèle observé, cela comporte des ambiguïtés dues à la sous observation des images 2D d'un corps en 3D et à la variabilité de l'apparence des personnes.

Dans la littérature, on peut dissocier ceux qui focalisent sur la reconstruction en partant d'une image unique de ceux qui s'intéressent au suivi de posture en analysant une séquence vidéo.

Dans le cadre d'utilisation d'images uniques, (Boulay 2007) ne traite pas la reconstruction de posture mais uniquement sa classification en un certain nombre de classes (debout, assis, penché, couché). Une première étape de détection de la silhouette par segmentation de l'avant-

plan est appliquée. La silhouette est ensuite analysée avec différents descripteurs (les caractéristiques géométriques, les moments de Hu, « *skeletonisation* », projections horizontales et verticales). La règle de décision utilisée dans la classification est la mesure de la distance par rapport aux descripteurs des modèles connus a priori. Le manque de robustesse de cette technique est dû au nombre faible de postures des modèles utilisés ainsi qu'aux descripteurs qui ne sont pas assez discriminants.

Dans (Agarwal & Triggs 2006) on trouve une reconstruction approximative de la posture en 3 dimensions. La silhouette est d'abord extraite. Ensuite, on calcule une régression non linéaire entre la posture et les descripteurs de la silhouette. Le descripteur utilisé est l'histogramme de contexte de forme : « ***Histogram of Shape Context*** ». L'apprentissage (pour la régression) est réalisé avec des RVMs : « ***Relevance Vector Machine*** ». Cette technique est originale. Par contre, l'approche n'a été validée que sur des postures de marche fronto-parallèles.

Dans le cadre du suivi de posture, (Poppe 2007; Moeslund et al. 2006) présentent une vue d'ensemble sur l'analyse du mouvement humain avec la vision.

Considérons à titre d'exemple (Brubaker et al. 2009) qui se place dans le contexte de personnes en marche. Les connaissances a priori sont utiles. Cela est exploité par l'introduction d'un modèle physique pour le suivi de personnes. C'est un modèle de squelette comprenant les équations dynamiques du mouvement. Il simule un marcheur anthropomorphique qui assure la cohérence de la séquence de marche lors d'occultations temporaires. L'inconvénient de cette approche est sa restriction à un mouvement de marche uniforme. Elle exclut les mouvements brusques ou imprévus.

(Fathi & Greg 2007) adoptent une technique de corrélation avec des exemples pour trouver la bonne configuration du corps humain. Leur technique utilise un échantillonnage de Gibbs et une descente de gradient.

Les problématiques intrinsèques à la vision monoculaire ne peuvent pas être levées par des avancées algorithmiques dans ce domaine. Les ambiguïtés sont généralement dues à des auto-occultations. La vision multi oculaire permet d'offrir une meilleure couverture de l'espace à reconstruire.

2.1.2 Estimation de posture par vision multi oculaire passive

L'intérêt d'une approche multi oculaire est de lever les ambiguïtés dues aux occultations. Il est impossible de reconnaître les parties non visibles depuis un point de vue unique. La triangulation permet de reconstituer les coordonnées en 3 dimensions d'un détail observé. Avec un système composé de plusieurs caméras, fixes et calibrées, il est possible de calculer la triangulation à chaque instant ***t***.

On peut répartir les approches en deux catégories selon l'utilisation ou non d'un modèle. L'approche sans modèle nécessite un apprentissage. L'apprentissage peut inférer directement la posture ou inférer des données intermédiaires permettant sa reconstruction.

Dans ce qui suit, on va aborder l'approche fondée sur le modèle du corps humain, le filtrage et la reconstruction par fusion. Ensuite, on décrit une approche fondée sur des exemples. Enfin, on présente une technique plus avancée fondée sur le modèle du corps humain reconstruit dynamiquement.

Approches fondées sur le modèle du corps humain

Les modèles géométriques utilisés pour la reconstruction de posture, sont divers. (Urtasun & Fua 2004) utilisent un modèle de squelette composé d'ellipsoïdes. (FONTMARTY 2008) utilise un modèle en cône tronqué pour développer un système modulaire réalisé pour une application

d'interaction avec l'Homme. Un système stéréoscopique mobile sur le robot a été utilisé en complément d'un système multi oculaire fixe dans l'environnement. (Sundaresan & Chellappa 2005) utilisent un squelette avec des formes super-quadratiques (surfaces de forme cylindrique arrondie aux extrémités).

(Sigal et al. 2004) utilisent un modèle graphique qui définit les relations entre les parties du corps. Ces parties du corps sont représentées par des cylindres qui se positionnent avec des paramètres continus en 6 dimensions à identifier (3 rotations, 3 translations). Ces cylindres sont projetés sur le plan de chaque image. Des descripteurs de contours à plusieurs échelles sont calculés pour identifier la vraisemblance de chaque hypothèse. La propagation de la croyance non paramétrée est utilisée avec un filtrage particulaire qui s'applique à un graphe. Cela permet de retenir les hypothèses qui répondent aux contraintes du graphe de tout le corps. C'est une approche « ***bottom-up*** ». Cette même approche de projection de cylindres sur le plan de l'image est utilisée dans (Deutscher & Reid 2005) sans modèle graphique, mais avec du filtrage particulaire combinant recherche hiérarchique et opérateurs de « ***crossover*** » inspirés des algorithmes génétiques.

Filtrage et suivi

Le filtrage fait partie de presque toutes les approches, mais avec des techniques plus ou moins avancées. Face à l'insuffisance de la précision des données perçues, le filtrage temporel est parfois considéré comme une issue pour pallier ce manquement. Beaucoup d'efforts se sont concentrés sur les modèles de mouvement pour contraindre le suivi. Au lieu de considérer plusieurs hypothèses de postures qui risquent de tomber dans des minima locaux, (Urtasun & Fua 2004) proposent un modèle temporel. Ce modèle temporel n'utilise pas directement les paramètres des postures, mais effectue d'abord une analyse des composantes principales. Le problème de suivi est ensuite appliqué sur les coordonnées issues des composantes principales. Une fonction de minimisation permet de choisir l'ensemble des points représentant la trajectoire qui engendre le moindre coût.

Fusion en volume voxellisé

Dans (Sundaresan & Chellappa 2005), les multiples images fusionnent en un nuage dense pour produire un volume de données (voxels). De 8 à 16 caméras sont utilisées pour obtenir ce nuage de voxels. Une approche « bottom-up » est combinée avec une approche « top-down » guidée par la connaissance du modèle et un algorithme de suivi. L'algorithme de suivi est un filtre de Kalman étendu itéré (IEKF : Iterated Extended Kalman Filter (Liang-qun et al. 2005)).

(Cheng & Trivedi 2007) reconstruisent aussi des voxels à partir de multiples images. Pour la reconstruction, ils proposent une technique probabiliste qui utilise un modèle de mixture de Gaussiennes multi composantes. Cela décrit la distribution spatiale des voxels. Chaque composante représente un segment ou un corps rigide et la collection des composantes est reconstruite avec des contraintes cinématiques correspondant au modèle de squelette prédéfini.

Approche fondée sur des exemples de postures

On retrouve dans (Rogez et al. 2008) une approche fondée sur les exemples qui est une alternative aux approches fondées sur modèle. Ils ont utilisé une variété géométrique (***manifold***) de dimension 2. On rappelle que la variété géométrique est un espace topologique qui peut s'apparenter à une généralisation de la classification. Les deux dimensions utilisées dans ce cas sont le point de vue et l'action. Cela permet de réduire la variabilité de l'apparence qui alourdit l'apprentissage à cause de ces deux facteurs. Des arbres de décision ont été utilisés pour retrouver la posture. Le descripteur testé à chaque nœud est un HoG « ***Histogram of Gradient*** »

(Dalal & Triggs 2005) appliqué aux images d'apprentissage qui ont été alignées. C'est-à-dire que leurs coordonnées ont été décalées pour que le centre du corps se trouve au même endroit pour toutes les postures. Des tests ont été réalisés sur des caméras fixes et en mouvement.

Il y a sûrement un compromis à trouver entre la complexité algorithmique et la force brute des moyens techniques récents. Ces moyens ont permis de lever certaines contraintes, comme le besoin de fixer les caméras, de les calibrer ou de les synchroniser. (Hasler et al. 2009) capturent les mouvements dans un environnement non structuré et avec de multiples caméras en mouvement. La synchronisation se fait avec le canal audio associé à chaque flux vidéo. La scène est reconstruite en 3D en identifiant la structure à partir du mouvement (***structure from motion***). Cela permet de reconstruire le modèle de l'arrière-plan statique en 3D tout en y identifiant la trajectoire de chaque caméra. (Urtasun & Fua 2004) justifient l'utilisation des modèles temporels par la mauvaise qualité des images. La qualité des images est une problématique associée aux moyens techniques utilisés qui évoluent au cours du temps. À titre d'exemple, (Hasler et al. 2009) utilisent des images en haute définition permettant la reconstruction de formes 3D texturées. Cela se fait à l'aide d'une approche spécifique au modèle de la personne en question qui est scanné en 3D pour alimenter l'a priori. Cette approche avec modèle a priori extrait la forme à partir de la silhouette (***Shape from silhouette***, ***Visual Hull***). (Cheung et al. 2003) ont appliqué cette technique à une personne en mouvement. Il est même possible d'augmenter le détail de la modélisation en représentant le corps de la personne comme une surface déformable. (Gall et al. 2009; Corazza et al. 2010; Li et al. 2011) relâchent la contrainte de l'existence des parties du corps rigides et ils optimisent le suivi de surfaces déformables avec reconstruction de la surface (***mesh***) en 3D de la personne et ses habits.

Discussion

Nous concluons à la fin de cette section, que les moyens techniques d'acquisitions utilisés (résolution des images, fréquence d'acquisition) influent grandement sur le choix de l'algorithme ainsi que sur le résultat final.

L'aspect multi oculaire améliore la précision des données et leur fidélité. Pour 2 caméras, on parle plutôt de vision binoculaire et les techniques utilisées sont spécifiques à ce type de dispositif. Si le nombre de vues est généralement entre 3 et 4, ce nombre n'a pas de justification théorique. Ce nombre peut être 3 ou plus et ce sont les moyens techniques et financiers qui le limitent. La qualité des données est quant à elle croissante en fonction de ce nombre. Jusqu'à 16 caméras ont été utilisées par (Sundaresan & Chellappa 2005).

Cette problématique du nombre de capteurs classiques ou passifs est minimisée lors de l'utilisation de capteurs actifs fournissant des données plus riches incluant la profondeur. C'est ce qui nous a amenés à les privilégier pour notre approche. Par ailleurs, comme les approches par labellisation sont plus robustes en vision passive, ceci nous a incités à les considérer pour notre approche.

2.1.3 Estimation de posture par mono capteur de profondeur et labellisation des parties corporelles

Avant de se lancer dans les détails de la littérature, on commence par introduire les capteurs de profondeur. Cela permet d'anticiper le potentiel technique à exploiter par les algorithmes d'estimation de posture.

Littérature propre aux capteurs de profondeur

Dans les approches précédemment décrites, beaucoup d'efforts se sont concentrés dans l'exploitation des riches informations contenues dans l'image, certains en essayant d'apparier les descripteurs entre les images donc ayant fortement besoin de textures, d'autres en utilisant les silhouettes ou des opérateurs de contours ou de gradients pour essayer de faire abstraction de ces textures variables. La profondeur est une abstraction géométrique qui nécessite une texture pour être calculée. Les capteurs actifs peuvent projeter eux-mêmes la texture. Une fois calculée, la profondeur est une autre façon de faire abstraction aux variations d'apparences. Des variations de forme persistent tout de même. Elles sont dues aux différences morphologiques entre les personnes et aussi à la large variabilité des styles vestimentaires.

La collaboration entre Plagemann et Ganapathi (Plagemann & Koller 2010; Ganapathi et al. 2010) se place dans le contexte de mono vision avec données de profondeur. Ils ne classifient pas les parties du corps, mais plutôt reconnaissent des points d'intérêt. Plus spécifiquement dans (Plagemann & Koller 2010) ce sont les extrema géodésiques de la surface qui correspondent aux points d'intérêt du corps humain qui peuvent être classifiés comme main, pied, ou tête avec des descripteurs locaux. L'approche permet aussi de détecter l'orientation 3D du point d'intérêt. Les points d'intérêt sont dénommés AGEX «***Accumulative Geodesic EXtrema***». Cette technique, décrivant des points épars, manque de robustesse comparé à la technique dense que l'on va décrire par la suite qui classifie la totalité des points capturés (totalité des pixels).

Dans la section suivante, nous allons décrire en plus de détails le travail réalisé par (Shotton et al. 2011a). C'est l'approche la plus pertinente connue lors de la réalisation de notre étude, car elle offre de bonnes performances comparées aux autres techniques existantes. Elle nous a inspirés pour la classification des parties du corps en utilisant des forêts de décision. Elle concerne la classification des pixels de surface de l'image de profondeur.

2.1.4 Travaux de Shotton et al.

Une particularité dans la détection de postures présentée par (Shotton et al. 2011a) est le non-usage d'information temporelle. Le post-traitement des données se limite donc au lissage des trajectoires. L'algorithme permet à lui seul une réinitialisation qui récupère le suivi à chaque nouvelle image.

La base d'apprentissage a été constituée uniquement à partir d'images de synthèse. La base de test réelle contient 8 808 images de profondeur labellisées à la main. Comme l'application est consacrée à un usage où l'utilisateur est face à la caméra, la rotation est limitée à ±120° dans la base d'apprentissage et la base de test.

On rappelle donc que l'objectif est de labelliser chaque pixel de profondeur à la partie du corps correspondante en utilisant le descripteur ci-dessous.

Pour ce descripteur, les caractéristiques utilisées dans l'image de profondeur se formulent de la sorte :

$$f_\theta(I,x) = d_I\left(x + \frac{u}{d_I(x)}\right) - d_I\left(x + \frac{v}{d_I(x)}\right)$$

Où $d_I(x)$ est la profondeur du pixel x dans l'image I, et le paramètre $\theta = (u, v)$ décrit les offsets u et v. La normalisation par $\frac{1}{d_I(x)}$ assure l'invariance par rapport à la profondeur vu que dans l'image de profondeur, la taille dépend de la profondeur (contrairement à notre approche qui a des données fusionnées en voxels dans l'espace cartésien, et qui n'a pas besoin d'une telle

normalisation). Cette normalisation permet une invariance à la profondeur, mais ne résout pas l'effet de la perspective.

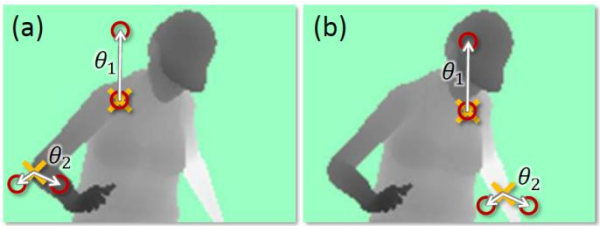

Figure 2-1 : caractéristiques de l'image de profondeur (a) grande différence entre les offsets (b) petite différence (Shotton et al. 2011a)**.**

Dans la figure 2-1, le pixel classifié est représenté par la croix jaune et les cercles rouges indiquent les offsets associés. Ces caractéristiques à elles seules représentent un classifieur faible qui une fois inséré dans une agrégation de classifieurs comme celui des forêts de décision permet de distinguer la partie du corps en question. Une forêt contient plusieurs centaines de milliers de ces classifieurs faibles. Le formalisme des forêts de décision est détaillé dans la section 3.3.4.

Le processus d'apprentissage et de classification avec des forêts de décision est assez générique. Il s'agit d'un apprentissage supervisé. Tous les pixels labellisés permettent de construire les arbres de décision. Cela se fait en itérant le défilement des données à travers les nœuds pour calculer des histogrammes. Ces histogrammes permettent de sélectionner la caractéristique à utiliser dans chaque nœud. Une fois l'arbre construit, les caractéristiques permettent de décider du prochain branchement dans l'arbre et les histogrammes au niveau des feuilles permettent de décider de la classe à affecter. Quand on a plusieurs arbres de décision (forêt de décision), la décision finale de la classe à affecter se fait par vote.

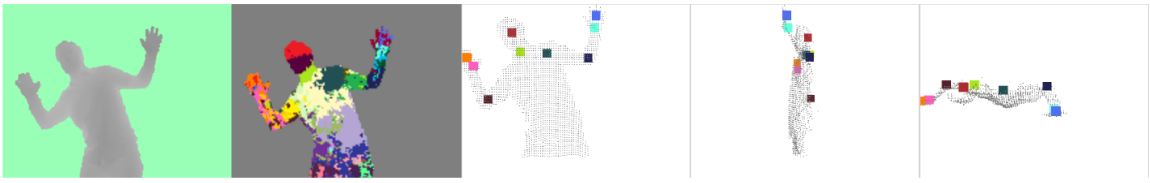

Figure 2-2 : Image de profondeur – Image classifiée en parties du corps – Joints vue de face – Joints vue de côté – joint vue de dessus (Shotton et al. 2011a)**.**

La figure 2-2 montre les résultats qualitatifs de labellisation des joints qui utilise une technique de recherche de modes basée sur le « mean-shift » (Cheng 1995) avec un noyau Gaussien pondéré. L'estimateur de densité (nombre de voxels par volume) par partie du corps est défini par :

$$f_c(\hat{x}) \propto \sum_{i=1}^{N} w_{ic}\, exp\left(-\left\|\frac{\hat{x}-\hat{x}_i}{b_c}\right\|^2\right)$$

Où $\hat{x}$ est une coordonnée dans l'espace du monde en 3D, N est le nombre de pixels dans l'image, w_{ic} est une pondération du pixel, $\hat{x}_i$ est la re-projection du pixel x_i dans l'espace étant donné la profondeur $d_I(x_i)$, et b_c est une largeur apprise pour chaque partie.

Les bases de données et l'infrastructure logicielle sont les ressources les plus contraignantes et nécessitent beaucoup de temps de développement. Ce n'est qu'une fois qu'elles sont disponibles qu'il est possible de tester et valider un algorithme. Nous remarquons que dans

plusieurs références citées, ces ressources ont été les mêmes et partagées par les coauteurs de ces références (Jamie Shotton, Ross Girshick, Min Sun, Jonathan Taylor, Andrew Fitzgibbon,…).

Parmi les extensions qui ont succédé à l'approche que l'on vient de décrire, on a d'abord (Girshicky et al. 2011). Sa principale contribution consiste à la régression directe de la position des joints. Il n'utilise pas de classe pour chaque pixel. La classe (bras, tête,…) est une représentation intermédiaire qui facilite le calcul des positions des joints, mais introduit une limitation. La régression permet de déterminer la position de joints même s'il n'y a pas de pixel correspondant à la classe en question. D'autres pixels assez proches peuvent contribuer au calcul.

L'apprentissage consiste à stocker dans chaque feuille d'arbre une distribution relative de l'offset 3D des joints concernés. Comme la distribution obtenue est trop complexe pour être représentée par une gaussienne, uniquement quelques vecteurs de vote relatifs sont représentés $\Delta_{ljk} \in \mathbb{R}^3$(avec l,j et k les indices des 3 coordonnées cartésiennes x,y et z), obtenus en gardant les centres des K modes (des positions des joints) les plus larges trouvés par « mean-shift ».

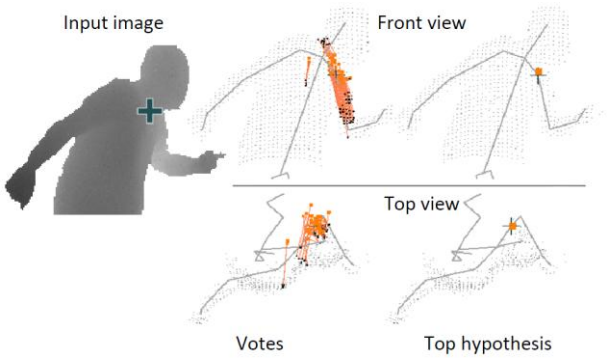

Figure 2-3 : agrégation des votes des pixels lors du test (Girshicky et al. 2011).

La figure 2-3 représente l'ensemble des vecteurs de votes des pixels en orangé, et le point central du résultat de ces votes. Les pixels des parties du corps voisines ont donc voté pour une partie du corps qui est occultée dans cette image. Cette technique a donc permis de retrouver les coordonnées d'un joint dont la partie du corps associée est occultée. La technique décrite précédemment n'aurait pas pu déterminer la position de ce joint puisqu'elle ne calcule que les positions des joints dont la partie du corps associée est représentée par des pixels.

L'approche de (Taylor et al. 2012) va au-delà de la régression de la position des joints, et s'intéresse à la régression qui permet de projeter chaque pixel de l'image de profondeur dans l'espace de coordonnées (u, v) du « vitruvian manifold » représenté dans la figure 2-4.

Figure 2-4 : The vitruvian manifold - (Taylor et al. 2012).

L'opération de correspondance s'effectue sur une posture initiale, cette étape est suivie d'une étape de correspondance qui permet de remonter à la posture observée dans l'image de profondeur.

Jusqu'ici, on a présenté l'apprentissage par forêt de décision, en supposant que toutes les variables soient indépendantes. Les variables sont les pixels à classifier en parties du corps d'une part et l'orientation de la personne de l'autre (l'orientation est considérée à titre d'exemple). En réalité, ces variables ne sont pas indépendantes. (Sun et al. 2012) profite de ce constat pour construire un modèle de régression par forêt de décision qui inclut une relation de dépendance entre les variables de sortie à travers une variable globale latente. Les variables latentes comme l'orientation du torse ou la hauteur de la personne peuvent augmenter considérablement la précision de la labellisation du pixel en partie du corps. La fixation de ces paramètres (orientation unique, hauteur de la personne) facilite l'apprentissage, car elle réduit la variabilité des données à apprendre. La complexité d'apprentissage est réduite.

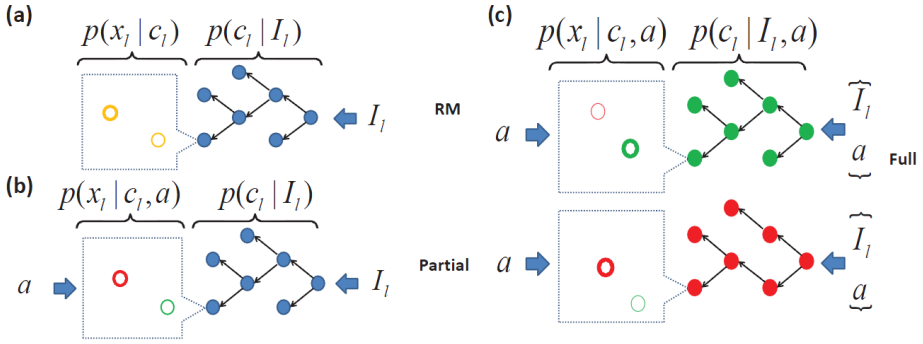

Figure 2-5 : Différents modèles de régression - (Sun et al. 2012).

On note par I les données de l'image, et L l'ensemble des votes des pixels l. On note par C l'ensemble des identifiants des feuilles c_l de l'arbre. Pour chaque joint j on calcule un score défini par :

$$S(x_j|I) = \sum_{l \in L} \sum_{c_l \in C} p(x_j, c_l|I_l) = \sum_{l \in L} \sum_{c_l \in C} p(x_j|c_l) p(c_l|I_l)$$

On dispose de deux méthodes de régression conditionnelle. Une méthode partielle implique le partage de la structure de l'arbre pour stocker dans les feuilles les différentes valeurs correspondant à chaque valeur de la variable conditionnelle. Ou une méthode totale, où on construit un arbre pour chaque valeur de la variable conditionnelle.

La figure 2-5 schématise la méthode classique ainsi que les deux méthodes de régression conditionnelles. Dans une méthode classique, le résultat est une pondération de tous les modes de la variable latente, alors que dans les méthodes conditionnelles, seule la valeur associée au mode en question contribue au vote.

Nous remarquons que toutes ces extensions de la technique de base (Shotton et al. 2011a) mettent en œuvre plus de moyens et plus de calcul pour raffiner le résultat de construction de posture, mais héritent tous des limitations associées à la mono vision (principalement l'auto occultation). C'est pour cela que l'on introduit à partir de la prochaine section l'intérêt de coupler plusieurs capteurs de profondeur.

2.1.5 Estimation de posture par multi-capteurs de profondeur

Notre approche se place dans ce contexte de détection de posture 3D avec multiples capteurs de profondeur. Vu que cette approche a été motivée par la disponibilité depuis fin 2010 des capteurs de profondeur à bas coût, on prévoit un gain d'intérêt croissant à cette approche. Par contre, un délai est nécessaire car ceci induit un renouvellement du matériel, et surtout de l'infrastructure logicielle qui a besoin d'un développement associé pour en faire une plate-forme. Il y a aussi des problèmes techniques à résoudre. Même s'ils ne présentent pas le centre d'intérêt de l'étude à mener, ils la ralentissent et influent le choix des solutions en fonction de leur faisabilité. Pour la synchronisation des capteurs par exemple (Berger et al. 2011) ont opté pour une solution mécanique pour améliorer l'interopérabilité, mais en contrepartie leur fréquence de fonctionnement a été divisée par quatre, ce qui la sort du cadre de suivi en temps réel.

Les deux références à l'état de l'art que l'on a constaté dans cet axe sont (Zhang et al. 2012; Berger et al. 2011). Si tous les deux fusionnent les données en voxels, ils reprennent des techniques classiques d'optimisation pour la reconstruction des squelettes. Cela leur a permis de réutiliser les algorithmes déjà existants et implémentés.

Les capteurs de profondeur apportent des données plus riches permettant de minimiser leur nombre comparé à des capteurs de vision classique. Ce nombre reste quand même limité par des phénomènes d'interférences dues aux mires projetées.

2.1.6 Récapitulatif et résumé de notre approche

Le tableau 2-1 récapitule les bases de données existantes les plus citées dans la littérature. On constate que l'utilisation de la profondeur est assez récente, car les références antérieures se contentent des flux de caméras passives. L'appariement de points et l'application d'algorithmes inférant des coordonnées 3D sont appliqués a posteriori comme c'est le cas de (Tenorth et al. 2009). Ils utilisent un algorithme de suivi pour reconstruire les mouvements. Cela a l'inconvénient d'être limité en précision (entre 5 cm et 10 cm) et en robustesse.

La seule base de données existante utilisant des données de profondeur est celle de (Shotton et al. 2011a). Par contre, comme elle est limitée à une application ludique, elle n'utilise qu'un seul capteur.

	HumanEva datasets (Sigal et al. 2010)	CMU MoCap Database[4]	TUM Kitchen (Tenorth et al. 2009)	(Shotton et al. 2011a)	Notre approche
Nb postures	80K	2605 séquences de plusieurs centaines de postures	36646 depuis 20 séquences[5]	500K Réelles + virtuelles	**23K**
Cameras	4 Couleurs 3 Noir et blanc	1 Couleur	4 Couleurs	Mono vue profondeur	**3 RGB-D**
Profondeur	Non	Non	Non	Oui	**Oui**
Multi Vue	Oui	Non	Oui	Non	**Oui**
Calibration	Oui	Non	Oui	Oui	**Oui**
Contenu des séquences	Marche, Course, Lance/Atrappe, Boxe	Marche, course, diverses actions	Actions dans une cuisine	Conduite, dance, frappe, course, navigation menu	**Mouvements divers, sports, chutes.**
Vérité terrain	MOCAP	MOCAP	Non, données extraites de la vidéo	Oui Synthèse + manuel pour le test	**MOCAP**
Disponibilité	Sur demande	Oui	Oui, sous demande	Limitée	**Sur demande**

Tableau 2-1 : Récapitulation des caractéristiques des bases de données disponibles.

Notre approche est d'utiliser plusieurs capteurs de profondeur pour calculer une fusion bas niveau et appliquer une étape intermédiaire qu'est la reconnaissance de chaque voxel et sa classification en partie du corps. Pour cette approche, on a besoin d'une base de données fournissant en même temps les données brutes de profondeur, ainsi qu'une vérité-terrain avec un système de MOCAP. La précision du système de MOCAP et sa calibration avec les données de profondeurs sont les conditions nécessaires pour nous permettre de générer la vérité-terrain de la labellisation des voxels pour toutes les postures. Ces données sont importantes pour l'apprentissage hors-ligne ainsi que pour la suite lors de l'évaluation quantitative de notre algorithme de classification et de notre algorithme global de reconstruction de postures. Il est aussi à noter que l'utilisation de données réelles (et non pas données synthétisées à partir de modèles virtuels) nous assurent un rapprochement des conditions d'utilisation.

Dans ce qui suit, on va présenter notre plate-forme multi capteurs, élaborée pour la création de notre base de données.

[4] http://mocap.cs.cmu.edu/
[5] https://ias.cs.tum.edu/software/kitchen-activity-data

2.2 Description de notre plate-forme multi capteurs

2.2.1 Réseau de capteurs

Pour reconstruire la posture de la personne, nous avons opté pour l'utilisation de plusieurs caméras de profondeurs pour pallier les inconvénients d'auto-occultation. Ce travail coïncide avec la démocratisation des capteurs de profondeur ainsi que leurs outils de développement. Notre ambition a été de faire appel à un réseau de caméras fournissant la couleur et la profondeur (même si dans notre cas, l'exploitation de la couleur se limite à la calibration et à la segmentation dans certains cas). Pour garantir le temps réel, ce réseau de caméras est connecté à un seul PC à travers le bus USB. L'infrastructure développée nous permet aussi d'avoir un réseau de machines connectées à travers un réseau TCP/IP. Le mode de fonctionnement en réseau est plus pertinent dans le cas de caméras distantes les unes des autres et dont les champs ne se recouvrent pas. Les caméras distantes qui ne filment pas la même personne en même temps ne fusionnent pas les données d'une même posture. Par contre, elles permettent de fusionner la trajectoire d'une personne se déplaçant dans un bâtiment à titre d'exemple.

Dans le travail qui va suivre, nous allons focaliser sur les caméras de profondeur dont les champs de vue se recouvrent, permettant ainsi la fusion en bas niveau de la même personne observée.

2.2.2 Notre environnement d'expérimentation

La reconnaissance des postures d'une personne a besoin d'une base de données à l'entrée de l'algorithme d'apprentissage. Cette base de données doit contenir le signal qui va être appris ainsi que la référence (ou la vérité-terrain) de ce à quoi correspond le signal observé. L'importance de l'approche que l'on a utilisée réside dans la segmentation en parties du corps. Plus de détails sur cette technique se trouvent dans le chapitre consacré à la reconstruction des postures. Pour le moment, on peut considérer que le signal à capturer consiste en de multiples canaux de profondeur, et le signal qui va constituer la référence terrain est délivré par les coordonnées des marqueurs fournies par le système commercial de MOCAP.

Tel qu'on l'observe dans la figure 2-6, le LAAS-CNRS est équipé d'un système de capture de mouvement (du fournisseur Motion Analysis appelé ici MOCAP) dont on voit un capteur en gros plan dans la figure 2-7.

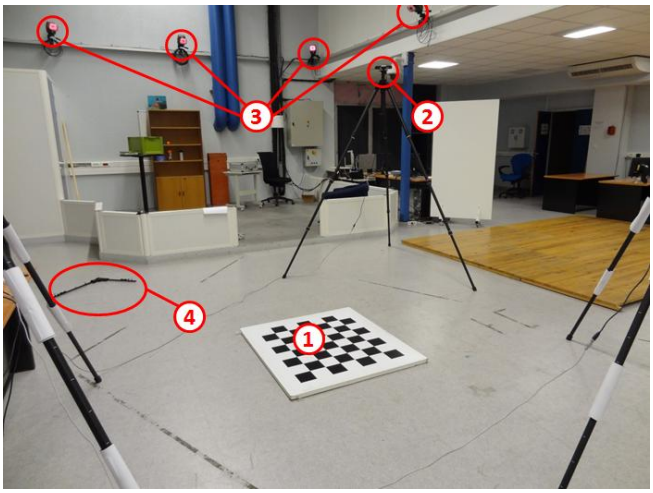

Figure 2-6 : 1 – Échiquier d'étalonnage Vision. 2 – Caméra de profondeur (xtion). 3 - Caméras IR du système MOCAP. 4 – L-Frame d'étalonnage du système MOCAP.

Figure 2-7 : Zoom sur une caméra infrarouge du système MOCAP.

Nombre de caméras Hawk	4
Résolution Hawk	640 x 480
Nombre de caméras Eagle	6
Résolution des caméras Eagle	2352 x 1728
Fréquence	200

Tableau 2-2 : Caractéristiques des caméras du système de MOCAP.

On présente dans le tableau 2-2 quelques caractéristiques du système de MOCAP indiquant la combinaison de caméras à faible résolution avec des caméras à haute résolution. La disposition des caméras est donc importante pour avoir une bonne couverture de l'espace.

Le processus de capture de mouvement est présenté en plus de détails dans (Bodenheimer et al. 1997).

Dans la figure 2-8 en plus du système de MOCAP, on perçoit l'installation des capteurs de profondeur. Deux types de capteurs de profondeurs sont disponibles dans le commerce. Ils sont présentés dans la figure 1-2. Pour choisir le nombre de capteurs, on a été amené à faire un compromis entre la quantité d'informations capturées et le taux d'interférence entre les capteurs. Il a fallu minimiser le nombre de capteurs qui projettent sur la même surface, car ces capteurs font de la stéréovision active. La disposition retenue est celle affichée par la figure 2-8.

On y observe en couleur rouge, vert et bleu, les trois capteurs d'images associés à chacun des capteurs de profondeur. Ils sont placés à 120° autour de la personne, sachant que leurs positions exactes sont déterminées par le processus d'étalonnage des capteurs d'images, qui sont fixes par rapport aux capteurs de profondeur.

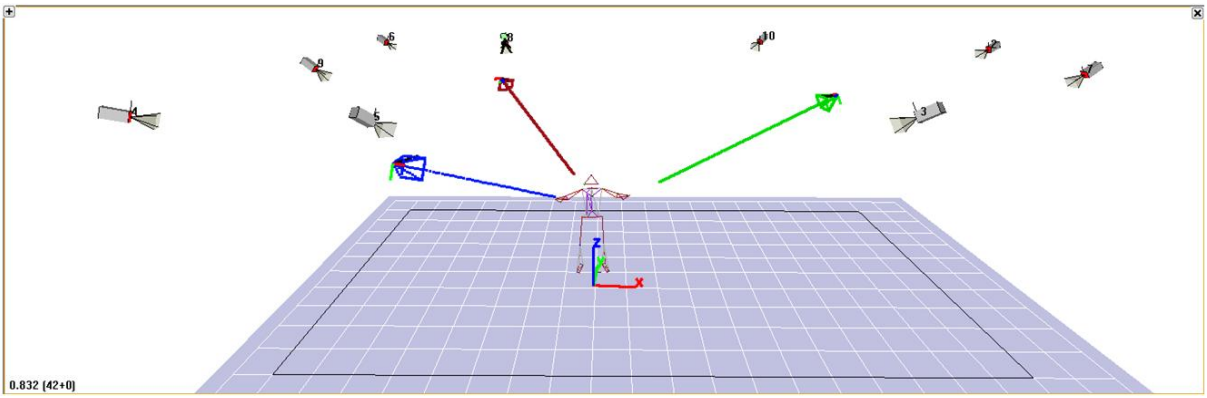

Figure 2-8 : Vue d'ensemble du correspondant virtuel de l'environnement de capture (1 segment = 20cm).

2.3 Acquisition des données

2.3.1 Acquisition

Les images RGB « Red, Green, Blue » et les images de profondeur sont acquises avec un xtion d'Asus. Les deux capteurs contenus dans les dispositifs kinect et xtion ont été conçus par le même fournisseur (Primesense) et ont un fonctionnement identique. Le choix a été tranché pour des raisons de connectiques, vu que le xtion n'a pas besoin d'une alimentation en plus de la connexion USB. Ce choix a entraîné des difficultés au niveau de l'utilisation du logiciel OpenNI qui va avec et qui n'est pas encore stable.

Les capteurs, répartis tel qu'on le voit dans la figure 2-8, observent des interférences dans des zones au sol où il y a une projection provenant de plusieurs capteurs. Cela n'a pas d'impact sur notre capture. Ce qui nous intéresse dans la scène, ce n'est pas les zones au sol, mais le volume de la personne. Le volume de la personne étant convexe, il est épargné des phénomènes d'interférences, car chaque zone du volume est illuminée par un seul projecteur du capteur en face. Ce phénomène reste donc marginal et a motivé le choix de cette configuration spatiale des capteurs.

2.3.2 Position des marqueurs pour la MOCAP

La figure 2-9 représente l'emplacement conseillé pour une capture avec 34 marqueurs. L'emplacement judicieux de ces marqueurs permet de constituer des repères pour chaque partie du corps, ces repères vont permettre de remonter à la meilleure approximation possible du centre de l'articulation et donc calculer une approximation de la matrice de transformation du modèle de squelette associé au corps humain. On parle ici d'approximations, car le modèle est sous-articulé et simplifié par rapport à la complexité des articulations du corps humain.

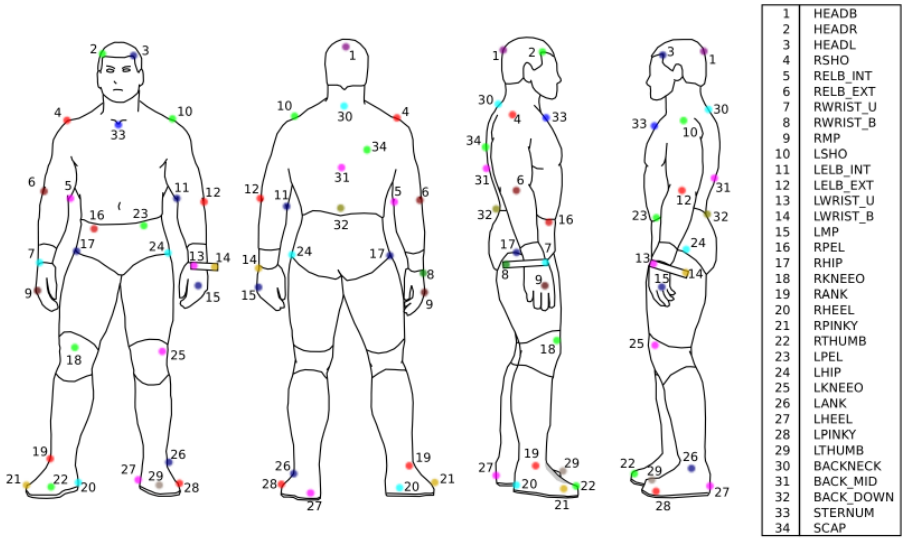

Figure 2-9 : Placement des marqueurs.

La figure 2-10 représente un squelette capturé avec le système.

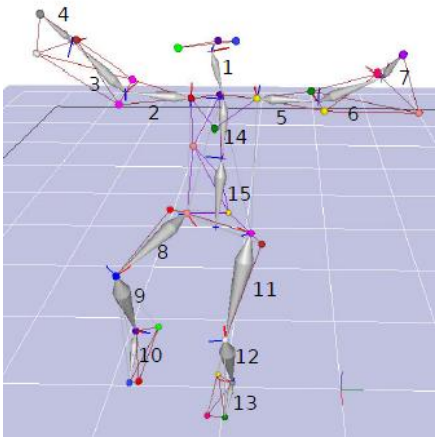

Figure 2-10 : Squelette du système MOCAP.

2.3.3 Codage et stockage des données

Une application a été développée pour la capture synchronisée de multiples canaux d'images et de profondeur provenant des capteurs (xtion ou kinect). Les contraintes de cette application sont :

- Le temps de traitement nécessaire au pilote pour capturer et segmenter les images.
- L'encodage des images couleur et profondeur.
- L'enregistrement sur le disque qui représente un goulot d'étranglement.

Une application « multi thread » a été développée :

- Un « thread » pour la capture.
- Un « thread » pour l'encodage et l'enregistrement dans une mémoire partagée.
- Un « thread » d'enregistrement sur le disque.

Le « thread » d'encodage représentait un goulot d'étranglement vu que l'encodage d'une image durait jusqu'à 30 ms. Comme on dispose de 6 images à encoder lors de chaque capture, cette opération a été parallélisée.

Le « thread » d'enregistrement sur le disque est resté l'unique goulot d'étranglement. Il a été allégé par sa parallélisation. En écrivant plusieurs fichiers en même temps on minimise le temps d'attente du « thread » global d'écriture.

2.4 Étalonnage géométrique et synchronisation temporelle

2.4.1 Étalonnage du système de MOCAP

La capture de mouvement a besoin d'un long processus qui commence par l'étalonnage statique. L'étalonnage statique permet de repérer les différentes caméras du système par rapport au même repère. C'est l'équerre d'étalonnage que l'on observe dans la figure 2-11 et aussi dans la figure 2-6-(4), qui représente le repère pour les caméras. Elle dispose de quatre marqueurs qui forment le repère du monde du système MOCAP.

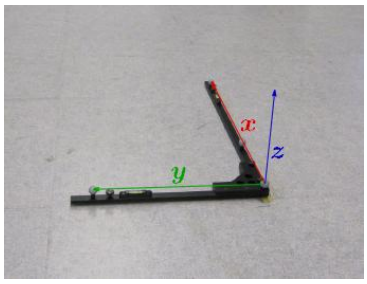

Figure 2-11 : Équerre d'étalonnage.

Ensuite, l'étalonnage dynamique permet de balayer tout l'espace de la capture pour affiner les paramètres intrinsèques de chacune des caméras. Cela va permettre de minimiser l'erreur de re-projection des marqueurs détectés dans l'espace de chaque caméra MOCAP.

2.4.2 Étalonnage des capteurs de profondeur

Vu que la position des capteurs de profondeur est fixe par rapport aux capteurs d'images couleurs, on a une fonction de transformation qui permet de re-projeter l'information de profondeur à partir du point de vue de l'image couleur. Cette transformation nous a permis de considérer le positionnement du capteur d'images couleur comme étant le même repère à l'origine de l'image de profondeur re-projetée.

L'étalonnage des capteurs consiste, grâce à cette astuce, en l'étalonnage classique des capteurs d'images couleurs.

2.4.3 Étalonnage des capteurs d'images couleur

L'étalonnage des positions des capteurs d'images couleur est donc un étalonnage classique, néanmoins une application a été développée pour faciliter le déroulement de la procédure, elle bénéficie notamment de :

- Détection automatique de l'échiquier et d'affichage à l'utilisateur en temps réel.
- Capture automatique des images d'étalonnage grâce à la détection de vitesse de déplacement de l'échiquier, ce qui permet d'éviter de capturer des images floues.
- Étalonnage intrinsèque et extrinsèque de toutes les caméras en même temps.
- Affichage des points 2D déjà considérés pour le calcul d'étalonnage afin que l'utilisateur puisse répartir équitablement les points à capturer sur l'ensemble de l'image.
- Affichage en temps réel de la position de toutes les caméras dans l'environnement virtuel.

2.4.4 Étalonnage géométrique du système multi capteurs RGB-D versus MOCAP

L'ajustement du système de vision avec le système de MOCAP consiste en une technique a priori rudimentaire mais qui a donné pleinement satisfaction en pratique.

L'équerre du système MOCAP est munie de clous de positionnement pointus et dont le placement est précis. Il a suffi de placer cette équerre dans le centre du repère de l'échiquier utilisé dans l'étalonnage des images couleur avec la bonne orientation. Vu que l'équerre MOCAP est précisément perçue par le système de MOCAP, il suffit donc de glisser légèrement sur le sol l'échiquier avec l'équerre MOCAP dessus afin de le placer au bon endroit (endroit où l'équerre

reprend sa position de référence et est affichée par le système MOCAP comme étant à la position (0, 0, 0) et à la bonne orientation). Les systèmes MOCAP et profondeurs sont donc maintenant étalonnés géométriquement. Cet étalonnage est quantifié lors de la re-projection des coordonnées des points capturés par le MOCAP directement sur les images couleur. Les marqueurs destinés au système MOCAP sont visibles sur les images que l'on enrichit avec des points indiquant leurs coordonnées capturées par le système.

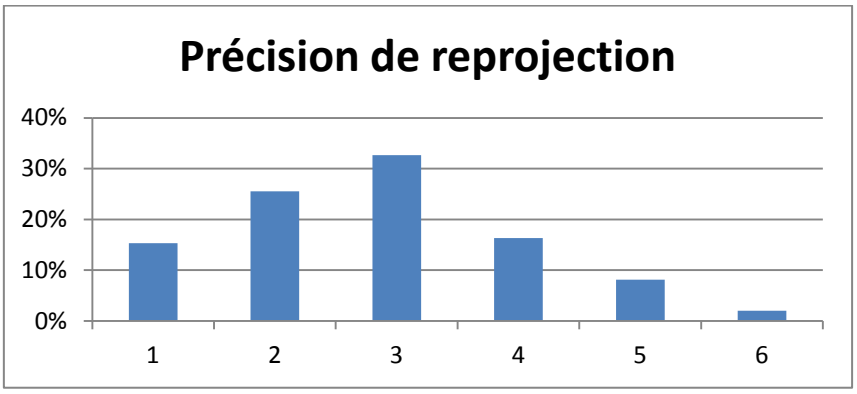

Figure 2-12 : Précision de re-projection (en pixels) des marqueurs sur les plans RGB des capteurs de profondeur.

La figure 2-12 indique les statistiques de re-projection calculées sur 100 points prélevés manuellement. La moyenne de l'erreur de re-projection est de 1.9 pixel.

2.4.5 Synchronisation temporelle : système MOCAP versus capteurs RGB-D

Les images RGB-D sont capturées chacune avec une référence temporelle provenant du capteur. Cette horloge relative à chacun des capteurs est assez précise. Pendant la capture d'une séquence, on néglige la dérive entre les horloges des systèmes de captures d'images et le système de MOCAP. Il reste donc à trouver l'écart temporel (« offset »), qui est constant et unique.

Vu que le système de MOCAP est cadencé à 200 captures par seconde, alors que les capteurs RGB-D peuvent varier entre 20 et 30, il y a entre 6 et 10 captures par image. On commence par re-projeter la position des marqueurs sur une image et vérifier à quel offset les marqueurs sont re-projetés au bon endroit. Comme lors des mouvements rapides, on a des différences de position allant jusqu'à 20 ou 30 pixels pour un marqueur visible d'une image à une autre, il suffit de faire varier l'offset parmi ses positions jusqu'à ce que le marqueur virtuel soit re-projeté bien au centre du marqueur visible. Cela permet donc de synchroniser les images avec le système de MOCAP. La précision est relative au système le plus fin qu'est le MOCAP (période de 5 ms).

On observe dans la figure 2-13 trois images couleurs successives sur lesquelles l'ajustement s'est réalisé avec la précision du MOCAP.

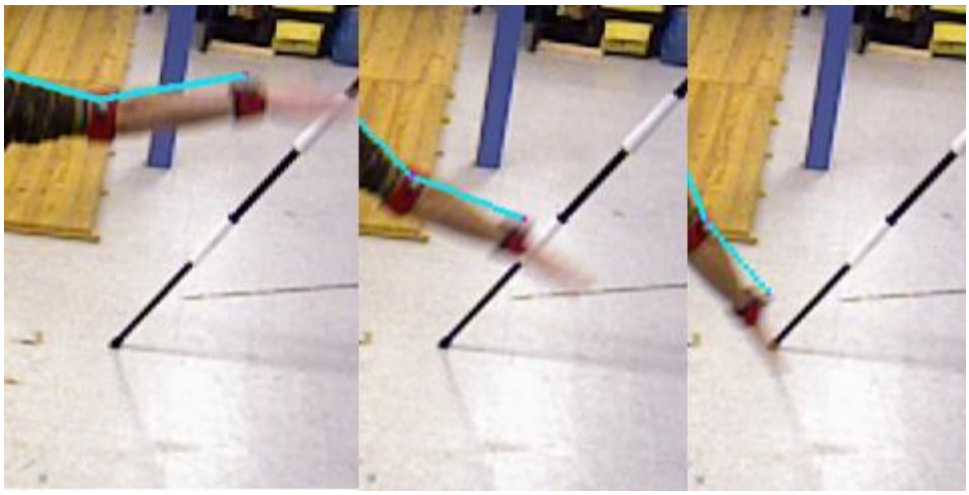

Figure 2-13 : Synchronisation du MOCAP avec un mouvement rapide.

2.5 Bases de données acquises

Nous avons constitué deux bases de données : NSC13 et IRSS35. Elles sont présentées dans le tableau 2-3. Ces bases contiennent des images RGB-D provenant de 3 sources (xtion) ainsi qu'une capture du squelette par le système de MOCAP.

La première, **NSC13**, est une acquisition effectuée avec 13 marqueurs indiqués dans la figure 2-14. Ces marqueurs sont placés sur des parties du corps proches des articulations et fixes par rapport à elles. Pour cette base de données, ce sont ces positions qui vont constituer les centres des parties du corps en question.

Le choix de ces articulations a été motivé par un compromis entre la simplicité de représentation et la fidélité de reproduction des postures. L'orientation de la main par exemple ou du pied n'influe pas dans notre application sur le sens que l'on donne à la posture.

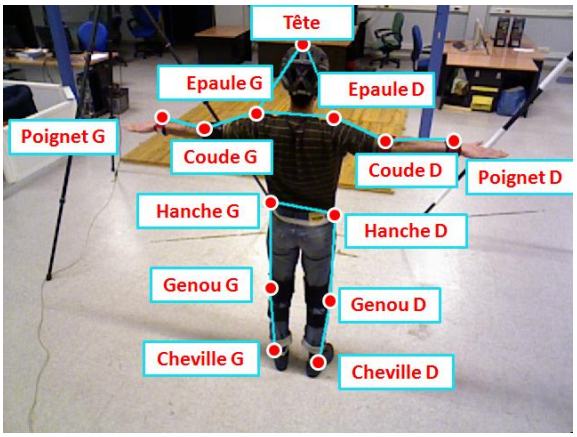

Figure 2-14 : Nom des marqueurs de la capture MOCAP.

Cette base a été enregistrée avec une fréquence moyenne de 5 images par seconde. Elle contient 5 groupements de séquences. Chaque groupement de séquences (référé simplement séquence) est un ensemble d'actions donc un ensemble de mini-séquences en soi.

	NSC13		IRSS35	
Vues Couleurs	3		3	
Vues Profondeurs	3		3	
MOCAP caméras	10		10	
MOCAP marqueurs	13		35	
Fréquence	5 images / s		20 images / s	
Nb séquences	5		8	
Total Nb Postures	1 951		21 569	
Séquences	M2	Posture en T, tour sur soi mouvements bras mouvements jambes	C1	Posture en T, mouvements bras, mouvements jambes, mouvements genoux, accroupis, bascule
	M3	Posture en T, marche, course, saut américain	C2	Posture en T, haltérophilie, tennis, volley ball, ping-pong, natation (bras), pétanque, lancement de poids
	M4	Posture en T, tour sur soi, pompes, break dance	C3	Posture en T, haltérophilie, tennis, volley ball, ping-pong, Natation (bras), Pétanque, Lancement de poids
	M5	Posture en T, tour sur soi, natation (bras), accroupis, chute arrière, chute avant, équilibre	C4	Posture en T, Tour sur soi, marche, course, assis debout, assis par terre, accroupi
	M6	Posture en T, ping-pong, natation (bras), volley ball, haltérophilie, Tennis	C5	Posture en T, Tour sur soi, marche, Course, assis debout, assis par terre, accroupi
			C6	Posture en T, saut américain, mouvements bras, équilibre, étirement, boxe, bowling, danse, chute avant, chute arrière, conduite, équilibre
			C7	Déplacer chaise, s'asseoir, balayer assis, déplacer meuble, balayer, bouger et filmer, jouer avec des balles
			C8	Posture en T, tour sur soi, mouvement articulations, équilibre, mouvement accroupi, karaté, échauffement, toucher les pieds, mouvements sur genoux
			C9	Posture en T, tour sur soi, haltérophilie, marche, tennis, volley ball, course, ping-pong, assis par terre, natation (bras), pétanque, lancement de poids, saut à la corde

Tableau 2-3 : Détails sur les bases de données acquises.

La deuxième séquence **IRSS35** a été effectuée avec la totalité des 35 marqueurs tels que présentés dans la section 2.3.2. Un post-traitement a permis de ressortir les positions des centres des articulations qui vont être la référence des centres des parties du corps. Sa cadence avoisine les 20 images par seconde. Elle possède également les 3 canaux d'images et de profondeur. Elle contient 8 séquences effectives (les autres sont des séquences d'étalonnage). Chaque séquence contient un ensemble d'activités humaines. À titre d'exemple, la séquence sport contient des actions d'haltérophilie, de Tennis (coup droit, revers, smash) et volley (passe, réception mains jointes, réception droite et réception gauche). Ces séquences incluent

intentionnellement des auto-occultations notables (accroupi, croisement des jambes et des bras). Le nombre total d'images, ou de captures, est de 21 569, produisant autant de postures.

Des vidéos des séquences acquises contenant les images profondeur et couleurs augmentées de la capture de mouvement sont accessibles via le lien URL :

http://www.wassfila.com/BPR

2.6 Conclusion

Dans ce chapitre, nous avons évoqué l'état de l'art de la capture de mouvement sans marqueurs. Nous avons couvert les différentes techniques pour justifier notre choix. Après avoir indiqué les limitations liées à la vision passive monoculaire et multi oculaire, nous avons justifié notre choix d'utiliser plusieurs caméras de profondeur. Par contre, la problématique associée est le manque de disponibilité de bases de données réelles pour ces capteurs RGB-D. Cela a motivé la création de notre propre base de données que nous avons présentée. Tout cela prépare l'introduction de la principale contribution qu'est l'algorithme de reconstruction de posture. Nous allons l'aborder dans le prochain chapitre et avec la vérité-terrain disponible pour notre base de données, nous allons pouvoir évaluer notre algorithme de façon quantitative mais aussi exploiter celle-ci pour l'apprentissage. Enfin une comparaison avec l'approche mono capteur de Shotton et al sera présentée au chapitre 4.

Chapitre 3 : Formalisation et implémentation de notre algorithme

Le but de notre algorithme est de reconstruire la posture en partant de données de profondeur. Notre approche privilégie des descripteurs 3D des parties corporelles contrairement aux travaux de travaux de Shotton et al.

Ce chapitre présente le synopsis de notre approche en comparaison avec celui de Shotton et al. Puis, nous détaillons le formalisme sous-jacent notamment les « Random Forest ». Son implémentation est décrite et notamment le réglage (« tuning ») des paramètres libres.

3.1 Introduction

La démarche est de fusionner les images de profondeur en un volume 3D occupé par la cible puis nous classifions les parties volumiques du corps sur la base de descripteurs 3D associés.

Supposant connus l'étalonnage géométrique de la plateforme multi-capteurs, la vérité terrain sur la trajectoire 3D des articulations (Cf. chapitre 2), notre approche repose donc sur :

- La définition d'un nouveau descripteur adapté à la voxellisation.
- L'implémentation d'une librairie d'apprentissage « Random Forest » optimisée et adaptée aux données voxellisées.

Ces points sont détaillés dans les sections suivantes.

3.2 Synopsis de notre approche

3.2.1 Synopsis

La figure 3-1 schématise le synopsis de notre approche. Chaque étape est décrite dans une section dédiée.

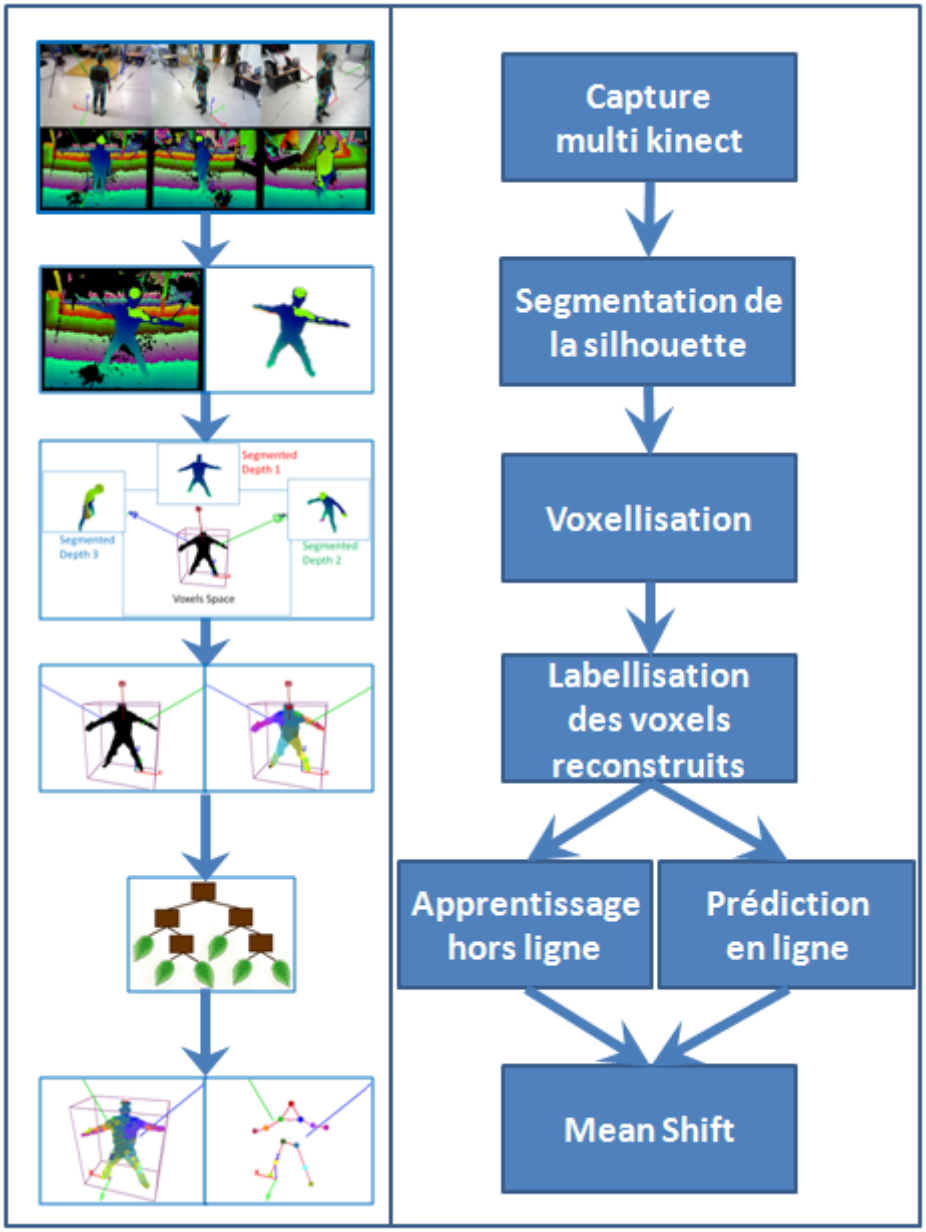

Figure 3-1 : Synopsis de notre approche.

Il s'agit tout d'abord de segmenter le premier plan contenant la silhouette de la personne dans chaque plan capteur. Ceci permet de voxelliser le volume en question. Ce volume est ensuite labellisé en s'appuyant sur la vérité-terrain du MOCAP commercial. Ces parties du corps labellisées automatiquement alimentent l'algorithme d'apprentissage hors-ligne. Cet apprentissage supervisé permet de classifier en ligne chaque voxel issu des nouvelles postures observées. Après labellisation des voxels, un algorithme de « Mean Shift » permet de localiser les articulations dans l'espace.

3.2.2 Notre approche vs. Shotton *et al.*

La figure 3-2 compare schématiquement notre synoptique avec celui de Shotton *et al.*

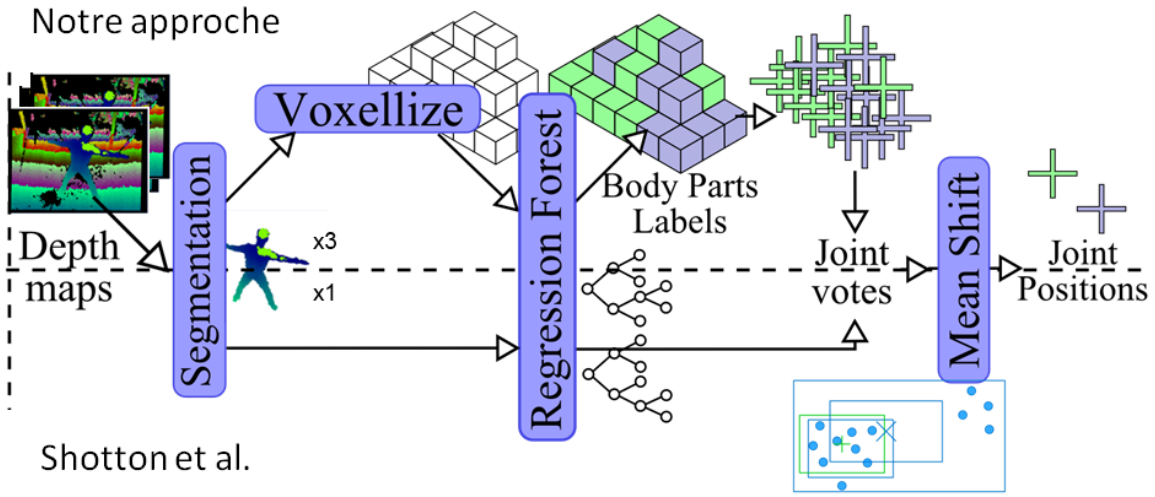

Figure 3-2 : Notre approche vs. Shotton et al. - (Filali et al. 2013).

Comme évoqué en préambule, notre approche se différencie sur le volet multi-capteurs permettant une voxellisation puis une classification des parties corporelles via des descripteurs 3D.

3.2.3 Étapes de traitement

Capture

Le processus de capture est détaillé dans la section 2.3 ainsi que dans la section *2.5*.

Segmentation de la silhouette

Le but ici est de segmenter la silhouette de la personne de l'arrière-plan dans chaque plan capteur. Nous pourrions privilégier les flux RGB et des techniques de segmentation par apparence ou par le mouvement. Pour s'affranchir de l'illumination et de toute hypothèse de mouvement, nous exploitons à l'instar de Shotton et al., le canal de profondeur.

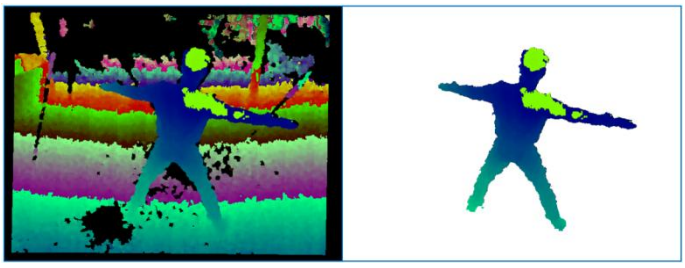

Figure 3-3 : Segmentation basée uniquement sur la variation de l'image de la profondeur.

La figure 3-3 illustre une segmentation sur un exemple issu de notre base de données.

Voxellisation

La voxellisation issue des divers points de vue (capteurs) est une étape clef de notre processus. Elle est illustrée par le rectangle central de la figure 3-4, le cube de l'espace voxellisé et les trois capteurs en couleurs différentes. Le rendu de cette image a été reconstitué à partir d'un point de vue virtuel. Les autres rectangles montrent l'image de profondeur déjà segmentée associée à chaque capteur.

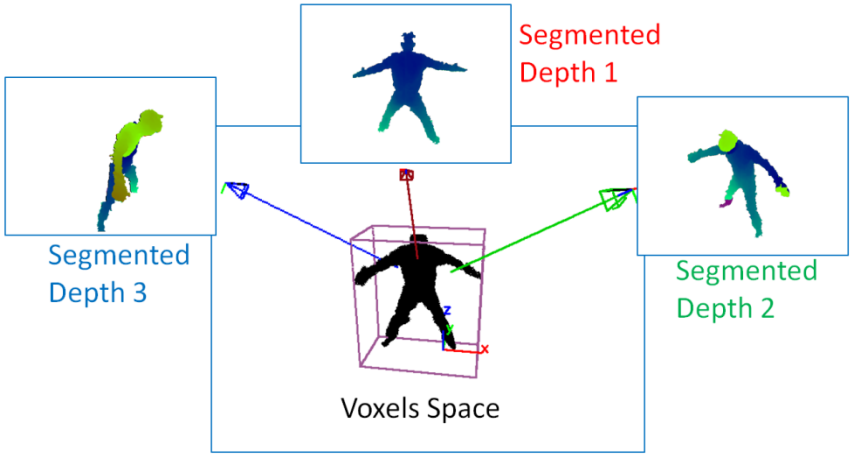

Figure 3-4 : Voxellisation d'une silhouette associée à une posture humaine.

Labellisation

La labellisation consiste à utiliser les données MOCAP pour labelliser les voxels associés aux parties corporelles. Ces parties corporelles sont au nombre de 13 pour la base de données NS-CAP13. La figure 3-5 illustre cette labellisation.

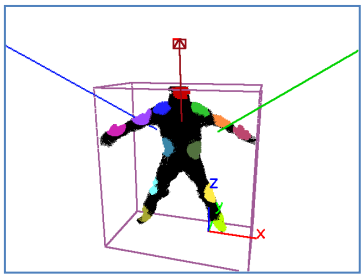

Figure 3-5 : Exemples de labellisation des parties corporelles.

Cette labellisation par MOCAP a un double intérêt : (1) générer des données pour notre processus d'apprentissage des descripteurs associés aux « random forests », et (2) constituer une vérité terrain pour évaluer les performances de reconstruction de notre algorithme.

Apprentissage et prédiction

L'apprentissage et la prédiction de la labellisation reposent sur une technique de « random forest » largement investiguée durant nos travaux.

L'apprentissage supervisé s'effectue logiquement hors ligne tandis que la prédiction se fait en ligne et donc sous contraintes temps réel.

Estimation de la position des centres des parties corporelles avec Meanshift

Une technique classique de « Meanshift » est utilisée pour estimer la position des centres des parties du corps grâce au nuage de voxels préalablement labellisés. Cette technique est rappelée ultérieurement.

3.3 Formalisation de notre algorithme

Cette section reprend les diverses étapes de notre algorithme et focalise sur les développements spécifiques, notamment la voxellisation et l'apprentissage dédié, enfin la localisation finale des articulations.

3.3.1 Voxellisation

La figure 3-6 illustre un élément de volume type voxel. Celui-ci peut être vide ou plein (grisé ou blanc dans la figure). Il lui sera associé dans notre cas un label de partie corporelle (tête, bras,…). Le voxel est repéré dans un référentiel local.

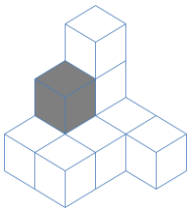

Figure 3-6 : Échantillonnage de l'espace et voxellisation.

La voxellisation est une méthode de discrétisation représentée par une matrice 3D binaire. Un objet voxellisé est alors une représentation d'un objet continu sur une grille de voxels régulière (Cohen-Or, D., and Kaufman 1995).

Voxellisation et caméras passives

La voxellisation est historiquement inférée sur des images 2D issues de caméras perspectives à l'instar de (Gond et al. 2008). Ils en extraient les données volumiques 3D mais ne classifient pas les parties du corps. Ils utilisent un descripteur unique pour décrire la posture à partir du volume. Cela implique une complexité combinatoire pour apprendre toutes les postures contrairement à un apprentissage dédié par partie corporelle. La démarche est similaire dans (Cheng & Trivedi 2007) mais les auteurs exploitent la base vidéo publique HumanEva (Sigal et al. 2010). Il existe néanmoins des travaux sur la voxellisation de données de profondeurs ; ceux-ci sont listés ci après.

Voxellisation et caméras actives

La figure 3-7 schématise la voxellisation issue de caméras passives ou actives.

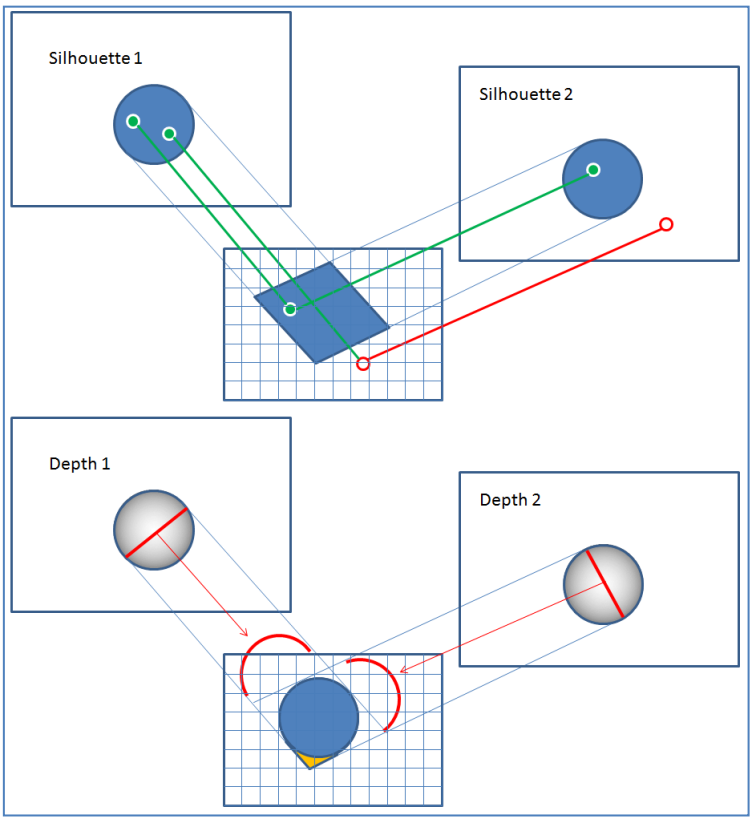

Figure 3-7 : Voxellisation par vision passive vs. vision active.

La voxellisation suppose que l'espace de travail est borné et représenté par un parallélépipède discrétisé en volume élémentaire (voxel). Chacun de ces voxels va ensuite être projeté sur le plan de l'image de coordonnées u et v. On considère ici le modèle de projection sténopé mettant en jeu les paramètres intrinsèques du capteur (Équation 1) notés fx, fy, cx et cy, s étant un facteur d'échelle. R et T représentent la localisation 3D du membre corporel dans le référentiel capteur.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [R \quad T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Équation 1 : Modèle de la caméra sténopé (pinhole)

Pour des caméras passives, classiquement, on projette un voxel sur la première image. S'il ne correspond pas à un pixel à l'intérieur de la silhouette, le voxel est considéré comme inoccupé et le test s'arrête. Sinon le test continue et le voxel est projeté sur le reste des images. Dans la figure 3-7, deux points de vue sont représentés.

Pour des caméras actives, en plus de ce test préliminaire, on compare la distance de ce voxel par rapport à la caméra avec l'information de profondeur. Cela permet d'éliminer plus de voxels que dans la voxellisation par silhouettes. La voxellisation, même pour des images de profondeur, peut donner lieu à des zones occultées e.g. la zone orange dans la figure 3-7.

Remarque
Une fois la voxellisation est calculée et la labellisation effectuée, les données sont prêtes pour l'apprentissage qui va alors s'appuyer sur le MOCAP commercial. Pour la suite, on dénomme échantillon une donnée relative à un voxel ; un descripteur est alors calculable sur chacun de ces échantillons.

3.3.2 Caractérisation des descripteurs de taille fixe

Chaque partie corporelle est supposée avoir un voisinage discriminant. Ainsi, pour une même posture, le voxel du centre de la tête a toujours un voisinage semblable, plein vers le bas, et vide au-delà d'une certaine distance dans le reste de la périphérie.

Chaque partie corporelle déjà segmentée représente une même classe à apprendre. Le principe est de classer sur la base des voxels voisins.

On appelle descripteur, le vecteur utilisé dans la fonction qui nous renvoie la classe du voxel après caractérisation de son voisinage.

Nous avons considéré successivement un descripteur de taille fixe puis un descripteur de taille variable.

Pour tout descripteur de taille fixe, on définit le test τ de l'espace voxellisé comme suit :

$$\tau(V;x,y,z) = \begin{cases} 1 \; si \; V(x,y,z) \neq 0 \\ 0 \; si \; V(x,y,z) = 0 \end{cases} \qquad (1)$$

Où V représente la fonction qui retourne l'identifiant de la partie du corps à partir des coordonnées 3D. Le résultat est 1 lors de la présence de « matière » versus 0 pour « non-matière ». Le descripteur est composé d'un ensemble de $\boldsymbol{n_d}$ vecteurs 3D qui définissent le test binaire. Le descripteur est défini par la chaîne de bits de dimension f_{nd} définie par l'équation :

$$f_{n_d}(V;P) := \sum_{1 \leq i \leq n_d} 2^{i-1}\tau\big(V;P_x + x_i, P_y + y_i, P_z + z_i\big) \qquad (2)$$

Dans l'équation (2), P représente le centre du voxel en question. Ces coordonnées 3D sont Px, Py et Pz. Xi, yi et zi sont les coordonnées du $i^{ème}$ vecteur du descripteur.

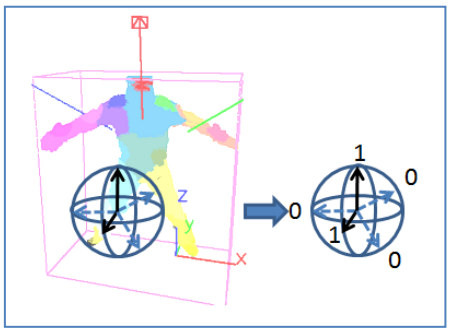

Figure 3-8 : Descripteur 3D de taille fixe.

La figure 3-8 illustre un descripteur pour $n_d = 5$. Chaque bit correspond à un vecteur relatif.

Pour ces descripteurs de taille fixe, détaillons ci-après le processus d'apprentissage puis de labellisation.

3.3.3 Classification à descripteur de taille fixe

Cette étude préliminaire exploite deux bases de données respectivement synthétique et réelle. La première repose sur des mouvements de marche provenant de la librairie CMU[6] et un modèle d'avatar provenant du logiciel « make human »[7]. La seconde provient de la base vidéo publique HumanEva (Sigal et al. 2010) ; celle-ci intègre 7 flux vidéo issus de caméras passives i.e. 4 caméras couleurs et 3 caméras achromes . Nous avons labellisé les parties corporelles de 56 images manuellement produisant 8 postures labellisées dont 7 ont été utilisées pour l'apprentissage et une pour le test.

Classification par Réseau de Neurones

En marge de nos investigations sur caméras actives, une étude préliminaire sur données HumanEva (i.e. caméras passives) a été menée via une collaboration avec le laboratoire LRPmip[8]. Elle s'inscrit dans le stage de Mlle H. Chabani co-encadrée par T.Simon chercheur au LRPmip à l'IUT Figeac et F.Lerasle sur une classification par réseaux de neurones. Il s'agit d'utiliser un réseau de neurones multicouches illustré par la figure 3-9 gauche qui montre une silhouette avec descripteur superposé. Le point à classifier est annoté « Reference pixel ». La valeur de chaque point du descripteur est définie selon si le point est dans la silhouette ou à l'extérieur. Ce descripteur constitue l'entrée du réseau de neurones. La sortie est la classe correspondant à la partie du corps.

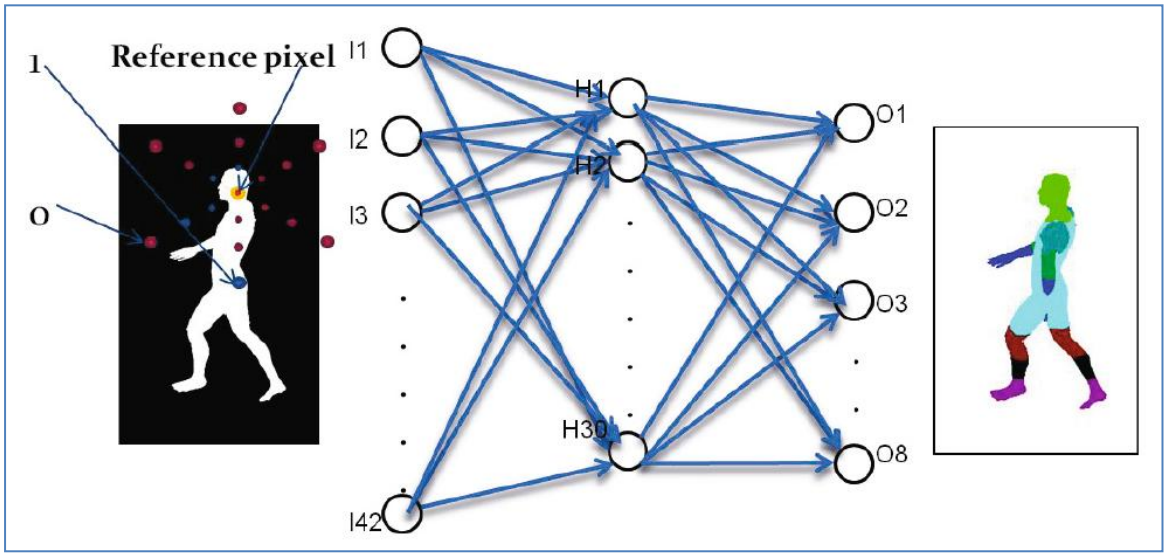

Figure 3-9 : Architecture du réseau de neurones (Chaabani et al. 2012).

Le réseau de neurones illustré ici est un réseau à trois couches perceptron « feed forward » connectées; il compte 42 éléments en entrée (I1,…I42), 30 éléments cachés (H1,…H30), et 8 éléments en sortie (O1,…O8) i.e. une par classe corporelle.

[6] http://mocap.cs.cmu.edu/
[7] http://www.makehuman.org/
[8] LRPmip-Université de Toulouse : UTM-IUT Toulouse 2 Figeac

Un total de 408 postures de la base de synthèse ont été utilisées avec 5 images 2D par posture (5 points de vue différents). 60 % des images ont été utilisées pour l'apprentissage, 20 % pour la validation et 20 % pour le test. Des optimisations de l'algorithme d'apprentissage ont été nécessaires pour réduire l'usage de la mémoire. Les données à apprendre ont été réparties sur 16 sessions d'apprentissage déroulées successivement. Notre réseau délivre une incertitude sur la classification de chaque pixel. Le résultat de classification des pixels dépassant le seuil de certitude est de 92 %. Ces études ont donné lieu à la publication (Chaabani et al. 2012).

Classification par k-PPV

La deuxième investigation à descripteur de taille fixe exploite une technique bien connue de k plus proches voisins (notée k-PPV) et évaluée sur des flux issus ici encore de caméras passives, indifféremment images de synthèse et images réelles.

Cette technique k-PPV dénommée en Anglais « ANN : Approximate Nearest Neighbor » est définie comme suit par (Muja & Lowe 2009) :

Définition
Étant donné un ensemble d'échantillons $\mathrm{P} = \{\mathrm{p}_1, \dots, \mathrm{p}_\mathrm{n}\}$ *labellisés* dans un vecteur d'espace X, un nouvel échantillon $\mathrm{p} \in \mathrm{X}$ s'appuie sur les échantillons dans P, qui sont les plus proches de q pour classifer p.

Le même auteur met à disposition une librairie en C++ qui permet de faire appel aux fonctions de recherche et d'approximation des K plus proches voisin avec un temps sous-linéaire[9] par rapport à une technique non approximative.

Les résultats expérimentaux obtenus sont les suivants :

Type de la base	Nb Postures apprises	Nb échantillons appris (voxels)	Temps de classification par voxel	Temps de classification par posture
Réelle (HumanEva)	7	750 K échantillons	23 us	1.8 s
Synthèse	300	27 M échantillons	1 ms	80 s

Tableau 3-1 : Temps de classification avec k-PPV (Base Synth-L300P10).

Le tableau 3-1 indique le coût CPU pour : (1) labelliser un voxel, et (2) labelliser tous les voxels d'une même posture. On remarque que même pour un faible nombre de postures apprises, ce temps de calcul ne satisfait nullement les contraintes temps réel propres à notre application.

Nombre de vecteurs	Rayon	Classification	Précision de la reconstruction
37	35 cm	48.5 %	12 cm
54	70 cm	51.5 %	7 cm

Tableau 3-2 : Classification et précision en fonction de la taille du descripteur (Base de données RHE-L7P1).

Le tableau 3-2 indique le résultat de la classification des voxels ainsi qu'après reconstruction, la moyenne de la précision obtenue. L'apprentissage repose sur 7 des 8 images réelles, la dernière constituant le test. Deux tailles de descripteurs ont été étudiées pour vérifier l'influence du voisinage dans la classification.

[9] http://www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN

Limitation des descripteurs de taille fixe

L'utilisation de descripteurs à taille fixe a vite montré ses limites. Cette technique nécessite l'extraction de la totalité des descripteurs pour toute la base de données. Avec une application non optimisée de première évaluation, 4 GO de mémoire pour l'application d'apprentissage n'ont pu contenir que les descriptions de 300 postures qui correspondent à la description de 27 millions de voxels, chacun était décrit avec 37 vecteurs.

Une autre limitation observée est le temps de classification qui atteint les 80 secondes pour une posture quand on en apprend une base de 300 postures et 27 millions d'échantillons.

Ceci a motivé une taille variable du vecteur descripteur en fonction des tests déjà effectués. Chaque combinaison de descripteurs serait adaptée à la classe à identifier.

Un arbre de décision repose sur la sélection préalable d'un vecteur (descripteur) discriminant par nœud. Cet apprentissage (sélection) est décrit dans les sections suivantes.

3.3.4 Formalisation des « random forests »

On considère la forêt d'arbres décisionnels une méthode faisant appel à un nombre variable de descripteurs. Sa topologie implique que chaque entrée à classifier parcourt un chemin différent et est évaluée avec un nombre différent de noeuds de l'arbre (chaque nœud contient un descripteur).

Notations

Définissons quelques notations sous-jacentes à la formulation des forêts de décisions telles que décrites par (Criminisi 2011).

Une entrée de la base de données est considérée comme un échantillon représenté par un vecteur $v = (x_1, x_2, x_i, \dots, x_d) \in F$, avec les x_i représentant des descripteurs. Le vecteur v peut représenter un échantillon type pixel, ou ici, type voxel.

Le nombre de descripteurs représente la dimension d de l'espace des descripteurs et dépend de l'application. Ici, il est borné par la résolution d'échantillonnage et par la taille de la zone du voisinage du voxel à considérer. Une fois échantillonné, cet ensemble va constituer les descripteurs qui vont être considérées lors du choix de **candidats.** Les candidats sont représentés par $\phi(v) = \left(x_{\phi_1}, x_{\phi_2}, \dots, x_{\phi_{d'}}\right) \in F^{d'} \subset F$, avec d'représentant la dimensionnalité du sous-espace et $\phi_i \in [1, d]$ représente la dimension sélectionnée.

Lors de l'implémentation, ϕ est considéré comme une base de données de vecteurs permettant la définition d'un descripteur de taille variable. On échantillonne les vecteurs et on les sauvegarde dans cette base. Cet échantillonnage se calculera avec les paramètres suivants :

$\boldsymbol{\varphi_A}$: Moyenne de la norme des vecteurs v de ϕ.

$\boldsymbol{\varphi_{min}}$: Norme minimale des vecteurs v de ϕ.

$\boldsymbol{\varphi_{max}}$: Norme maximale des vecteurs v de ϕ.

$\boldsymbol{\varphi}$ peut être échantillonnée de la façon la plus triviale dans l'espace des coordonnées cartésiennes du vecteur v, paramètre par paramètre (x,y puis z), méthode dénommée UniCoord. L'approche retenue est celle où on échantillonne d'abord la norme du vecteur, ensuite sa

direction. La direction est échantillonnée uniformément sur une sphère, méthode dénommée UniNorm.

La génération de l'arbre est assimilée à un ensemble de tests organisés hiérarchiquement. Chaque test est modélisé par un nœud de l'arbre. Le test est une fonction de découpage dans l'espace des descripteurs. Il est aussi appelé classifieur faible, à l'image du « Boosting ». On représente la fonction de test d'un nœud j comme fonction à sortie binaire :

$$h(v, \theta_j) : F \times \mathrm{T} \rightarrow \{0,1\}$$

Avec $\theta_j \in \mathrm{T}$ représentant les paramètres (ayant pour valeur 0 ou 1) de la fonction de test du nœud j. Le point v qui arrive au nœud est envoyé au descendant gauche ou droit en fonction du résultat de la fonction de test.

On appelle point d'entrée de la base d'apprentissage, la paire (v, y) respectivement le vecteur d'entrée (associé à un voxel) et la sortie représentant son label (ici de la partie corporelle associée). On appelle base d'apprentissage S_0 l'ensemble des échantillons injectés à l'entrée de l'arbre au premier nœud d'indice 0. Cette base va être répartie tout au long de l'arbre. Pour chaque nœud d'indice j, S_j va se répartir entre les nœuds fils gauche et droit dénommés S_j^G et S_j^D, avec les notations :

$$S_j = S_j^G \cup S_j^D,$$

$$S_j^G \cap S_j^D = \emptyset$$

Lorsque cette base de données $\mathbf{S_0}$ ou $\mathbf{S}$ est constituée de données redondantes (deux postures de la base de données sont semblables), un facteur de sous-échantillonnage $\boldsymbol{\alpha}$ est appliqué pour équilibrer les données à l'entrée de l'algorithme.

$$\alpha \in [0\,,1]$$

Représentations

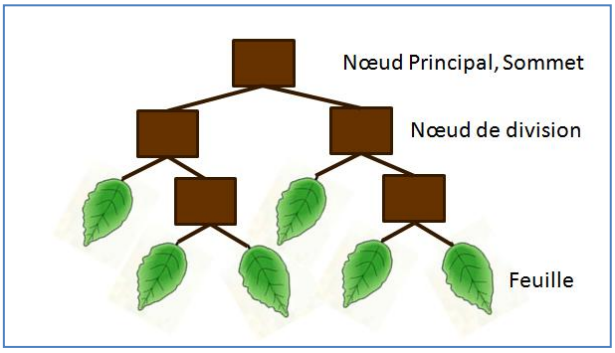

Figure 3-10 : Structure et terminologie d'un arbre de décision.

La figure 3-10 illustre un arbre de décision et la terminologie associée :

- Nœud principal, aussi nœud sommet ou sommet de l'arbre référencé par S_0.
- Nœud de division, appelé aussi bourgeon lorsque l'arbre est en processus de développement.

- Nœud feuille, ou feuille de l'arbre, est un nœud spécifique qui contient des statistiques complémentaires calculées lors de l'apprentissage et aboutissant à une labellisation de tout échantillon classé dans cette feuille.

L'arbre est ici représenté de haut en bas pour une cohérence avec les représentations classiques en littérature de telles structures.

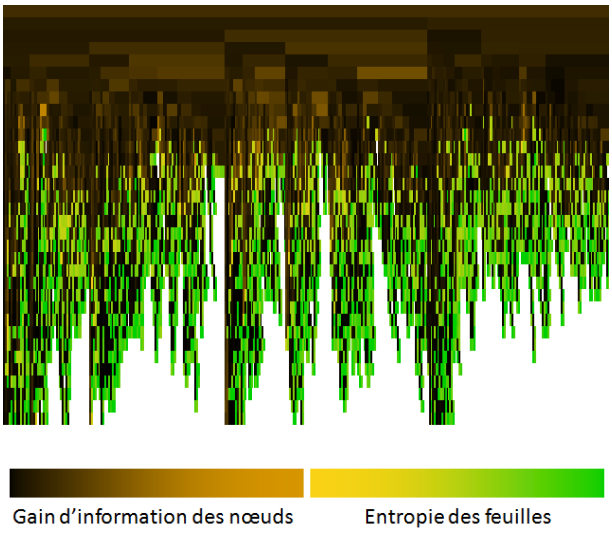

Figure 3-11 : Affichage d'un arbre de décision.

La figure 3-11 illustre un exemple d'arbre de décision qui peut être visualisé au fur et à mesure de son évolution lors de l'apprentissage. Les couleurs indiquées par les légendes donnent un aperçu visuel rapide de la qualité de l'apprentissage en fonction du gain d'information. On distingue aussi un aperçu visuel de la densité de l'arbre en fonction de sa profondeur, mesure que l'on représente également avec des courbes dans la figure 3-20. Les notions de gain d'information et d'entropie affichés sont définies ultérieurement.

Random forest

Une forêt de décision est constituée par ***N*** arbres de décision. (Breiman 2001) définit une forêt de décision comme un classifieur constitué d'une collection de classifieurs structurés en arbre $\{\mathrm{h}(v, \Theta_\mathrm{k}), \mathrm{k} = 1, \dots\}$ où $\{\Theta_\mathrm{k}\}$ sont des vecteurs indépendants et identiquement distribués. Chaque arbre dispose d'un vote unitaire pour la classe du vecteur v.

D représente la profondeur de l'arbre, l'origine 0 correspond à la racine de l'arbre.

Classification

La labellisation d'un échantillon représenté par son vecteur v consiste à parcourir l'arbre en commençant par le nœud sommet de l'arbre. En fonction du résultat du test $h(.,.)$ associé au nœud, l'échantillon va être dirigé sur le fils gauche ou droit du noeud. Ce processus est itéré jusqu'à l'atteinte d'une feuille de l'arbre qui n'a plus de fils selon un critère de gain d'information. À cette feuille est associé un histogramme indiquant la probabilité d'appartenance de l'échantillon à chacune des classes des parties corporelles. Cet histogramme constitue une distribution *a posteriori* : $p(c|v)$.

Il est possible de combiner les résultats des différents arbres de différentes manières. Nous avons comparé deux techniques : la pondération et le vote.

La pondération :

$$p(c|v) = \frac{1}{N}\sum_{n=1}^{N} p_n\,(c|v)$$

Pour la pondération, on fusionne d'abord tous les histogrammes des N arbres.

Le vote :

$$v = \max_{n}\left(\max_{c}(u)\right)$$

u représente la valeur de la case ***c*** de l'histogramme de l'arbre ***n***.

Pour le vote, chaque arbre vote en choisissant la classe dont l'occurrence de la classe est la plus grande. Ensuite, le vote est étendu à l'ensemble de la forêt.

Ces deux techniques sont implémentées puis comparées en section 3.5.8.

Apprentissage

L'apprentissage consiste à trouver le test $h(v,\theta_j)$ à associer à chacun des nœuds en optimisant une fonction objectif. On essaie de trouver la meilleure fonction qui répartit $S_j\ en\ S_j^G\ et\ S_j^D$ avec :

$$\theta_j^* = \arg\max_{\theta_j \in \mathrm{T}} I_j$$

$$I_j = I(S_j, S_j^G, S_j^D, \theta_j)$$

$$S_j^G = \{(v,y) \in S_j | h(v,\theta_j) = 0\}$$

$$S_j^D = \{(v,y) \in S_j | h(v,\theta_j) = 1\}$$

Ensuite, on itère le processus pour chacun des nœuds fils obtenu. La structure de l'arbre va dépendre de la façon dont on arrête la croissance d'une branche i.e. la transformation d'un nœud en feuille. Ceci se produit quand le critère d'arrêt ***C*** est satisfait.

$$C = C_D \vee C_I \vee C_m$$

C_D : Un critère sur la profondeur maximale D. Mais ceci est une approche déconseillée, car elle arrête arbitrairement la construction de l'arbre et induit des propriétés statistiques des nœuds tronqués différentes des autres.

C_I : Un critère sur le gain d'information que l'on développera par la suite nommé C_I.

C_m : Le nombre minimal d'échantillons ***m*** qui arrivent à un nœud, s'ils sont considérés comme peu nombreux pour contribuer à des études statistiques.

$$C_m = (S_j < m)$$

Regardons en détail la fonction de test (classifieur faible) de chacun des nœuds. Elle est définie de façon générique par $\theta = (\varphi, \psi, \tau)$ où ψ définit la primitive géométrique qui permet de séparer les données, τ représente les seuils dans la règle de décision (gauche-droite) et ϕ la fonction qui sélectionne quelques descripteurs du vecteur v.

La fonction objectif ou fonction coût nous permet de sélectionner le meilleur descripteur pour chaque nœud parmi ceux qui sont échantillonnés. Le choix de cette fonction objectif est un paramètre déterminant qui va influer sur la propagation des données dans l'arbre. Il y a des

modèles d'énergie classiques comme celui que nous allons voir, qui permettent d'avoir une fonction objectif classique, mais il est aussi possible d'adapter cette fonction objectif à un problème en particulier.

Le modèle d'énergie se fonde principalement sur deux critères, l'***entropie*** et le ***gain d'information***.

On utilise l'entropie de Shannon définie par :

$$H(S) = -\sum_{c \in C} p(c) \log\big(p(c)\big)$$

Où S représente un groupe d'échantillons d'apprentissage qui ont atteint une feuille. La variable C représente l'ensemble des classes, ici les labels corporels. La variable $p(c)$ représente la distribution empirique tirée de l'ensemble S.

En sélectionnant une fonction de test, on va obtenir deux groupes de données à gauche et à droite qui devraient être plus purs, cette amélioration de sélection peut être quantifiée par le gain d'information défini par :

$$I = H(S) - \sum_{i \in \{G,D\}} \frac{|S^i|}{|S|} H\big(S^i\big)$$

Ce gain d'information ***I*** va être calculé pour chaque nœud généré et donne lieu au critère d'arrêt C_I :

$$C_I = (I < t)$$

t est un seuil fixé par l'implémentation et discutée en section 3.5.5.

3.3.5 Récapitulatif des paramètres libres et variables

Pour faire le lien formalisation/implémentation, le tableau 3-3 récapitule les principaux paramètres libres et variables.

Symbole	Dénominations du paramètre
v	Vecteur, voxel, Échantillon, Caractéristique, Descripteur (Un descripteur est un vecteur dont l'origine est le voxel échantillon et qui pointe sur un voxel voisin)
y	Label, Classe, Partie du corps
φ	Ensemble de descripteurs candidats ϕ(v)
φ_A φ_{min} φ_{max}	Moyenne, min et max de la **norme** des vecteurs de la base ***φ*** pour le voisinage du voxel
d'	Dimensionnalité du sous-espace de ***φ*** Nombre de descripteurs candidats par nœud, ces candidats sont sélectionnés dans ***φ***

N	Nombre d'arbres de la forêt de décision
D_{MAX}	Profondeur maximale de l'arbre de décision
C_D	Critère d'arrêt de la croissance de l'arbre pour D = C_D.
S ou S_0	Base d'échantillons d'apprentissage injectée dans l'arbre
α	Facteur de sous-échantillonnage de la base ***S*** des échantillons
H	Entropie Entropie des feuilles (calculée pour les histogrammes des feuilles) ⇨ Indicateur de la qualité de sélectivité des feuilles
I	Gain d'information durant la génération de nœuds ⇨ Indicateur du pouvoir discriminant des nœuds
t	Seuil fixé sur le critère du gain d'information pour arrêter la croissance de l'arbre
m	Seuil fixé sur le critère du nombre minimal d'échantillons par nœud

Tableau 3-3 : Récapitulation des paramètres/variables des « Random Forests ».

Ces divers paramètres libres seront explicités dans la section 3.5.

3.3.6 Estimation de la position des centres des parties corporelles

Les voxels des parties corporelles à apprendre sont limités un rayon de 10 cm (Cf. section 3.4.2). Cette règle résulte d'un compromis entre nombre minimal de voxels à considérer et dimensions des parties corporelles.

Pour estimer les positions d'une articulation, une approche naïve est de calculer le barycentre des voxels labellisés dans cette classe. Ce calcul est peu robuste aux voxels mal classés comme illustré sur la figure 3-12 : le signe X représentant le barycentre a été influencé par le groupement d'échantillons à droite de l'image, alors que le signe + issu du « Mean shift » n'a pas été influencé par les échantillons mal classifiés tout à droite du volume total.

L'algorithme 3.1 du « Mean shift » consiste à refaire plusieurs itérations jusqu'à ce que l'on converge vers la zone d'intérêt correspondant à la localisation du centre de la partie du corps. Cette zone d'intérêt a été marquée par le signe '**+**' dans la figure 3-12.

Algorithm 1 Meanshift 3D Algorithm

```
for all body parts do
    repeat
        NewCenter ← ScanBoxForCenter()
        NewPartVolume ← ScanBoxForVolume()
        prevRatio ← Ratio
        Ratio ← NewPartVolume ÷ Box.Volume
        NewBox.Volume ← Ratio × Box.Volume × DataScatterRatio
        NewBox.Side ← cubicroot(NewBox.Volume)
        if NewBox.Side < minBoxSide then
            NewBox.Side ← minBoxSide
        end if
        Box ← UpdateBox(NewCenter,NewBox.Side)
    until (Ratio > MaxRatio) or (Ratio < prevRatio) or (NewPartVolume < minNbVoxes)
end for
```

Algorithme 3.1 : Mean shift

L'algorithme « Mean shift » s'applique à nos données 3D. Il s'initialise par une zone englobant l'espace voxellisé. Cette zone représente la fenêtre glissante tridimensionnelle de taille variable.

La première itération consiste à calculer le barycentre de cette fenêtre. Cette fenêtre va ensuite évoluer itérativement pour s'adapter à la taille des données. Le nombre de points en fonction de la taille de la fenêtre renseigne sur la compacité (concentration) des données, on utilise ce paramètre comme critère d'arrêt. La nouvelle taille de la fenêtre calculée est plus petite et elle assure une concentration croissante de la classe en question. Les erreurs de classification ou « outliers » sont alors moins influentes dans l'estimation du centre.

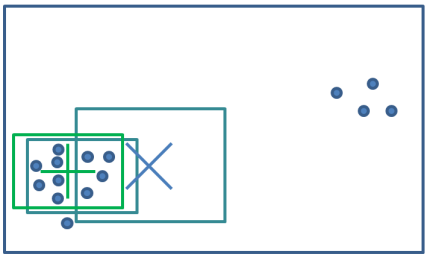

Figure 3-12 : Principe de l'algorithme Meanshift.

Dans le meilleur des cas, l'algorithme s'arrête quand la concentration de la classe atteint dans la fenêtre le seuil défini. Dans d'autres cas, pour éviter la divergence, l'algorithme s'arrête si la concentration baisse. Cela indique que l'hypothèse de compacité des données n'a pas été vérifiée. Il s'arrête aussi quand la taille de la fenêtre dépasse le seuil minimal défini.

3.4 Implémentation

Notre implémentation vise tout d'abord à étudier l'influence des paramètres libres pour fixer leurs valeurs. Celle-ci a été réalisée grâce à une librairie développée en C++ dénommée BPR[10]pour « Body Posture Recognition ». Elle est composée d'une vingtaine de classes et d'environ 14000 lignes de code.

Les paramètres libres à étudier sont relatifs à la voxellisation et labellisation des voxels.

3.4.1 Paramètres libres et voxellisation

Dans notre contexte applicatif, la discrétisation de la scène i.e. la taille des voxels est fixée à 1cm. Ce choix est motivé par une taille de voxel moyenne qui, une fois projetée dans l'image, correspond à un pixel.

Pour des considérations de coût CPU et mémoire, on ne calcule pas la voxellisation de toute la scène. Une scène de 2 m^3 contient 8 millions de voxels alors que la silhouette humaine 3D s'inscrit dans un rectangle de 800 000 voxels en moyenne. Comme la position de la zone d'intérêt est inconnue a priori, la démarche est d'évaluer quelques voxels choisis aléatoirement dans la scène. La figure 3-13 illustre le tirage aléatoire de 10 000 voxels affiché sur 3 vues ainsi que la boite englobante résultante.

[10] http://www.openrobots.org/wiki/bpr/

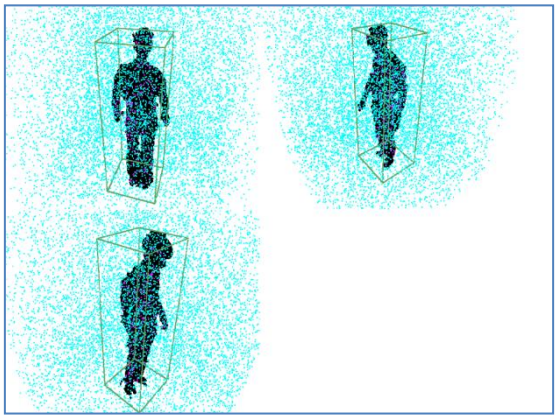

Figure 3-13 : Nuage de 10K voxels tirés aléatoirement.

À partir des voxels évalués en tant que « matière », on détermine la boite englobante pour laquelle on procède à l'évaluation de tous les voxels. Le coût CPU de la voxellisation grâce à cette approche a été divisé par 10.

3.4.2 Paramètres libres de la labellisation des voxels

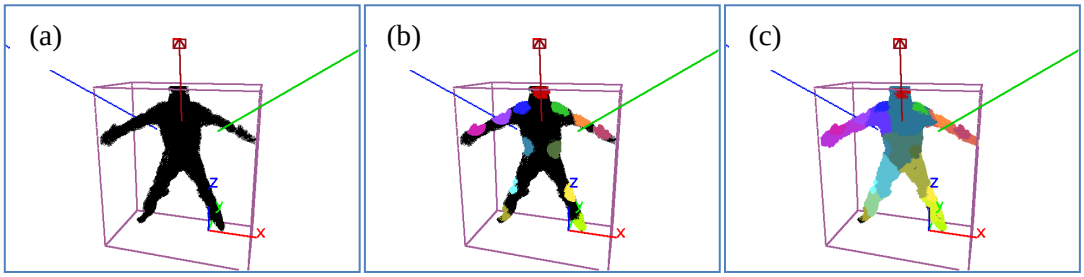

Figure 3-14 : Labellisation des parties du corps de référence (a) = volume non labellisé. (b) = parties du corps principales. (c) = parties principales et intermédiaires.

Les 13 parties du corps que l'on peut observer dans la figure 3-14 (b) sont considérées comme les parties du corps principales que l'on veut identifier. Pour pouvoir remonter à la position d'une partie à partir du volume voxellisé puis labellisé, on a intérêt à ce que ces parties soient identifiées de façon précise avec une faible erreur en distance. C'est pour cela que les zones labellisées de ces parties vont avoir une taille de rayon fixée à 10 cm. Cette taille est un compromis pour avoir un nombre de voxels minimal et des parties corporelles pas très grandes, donc homogènes.

Si l'on garde une seule partie qui constitue le reste du corps, elle serait plus difficile à apprendre. On a donc segmenté l'espace restant en 12 parties intermédiaires. La segmentation des parties intermédiaires part d'un point qui se situe à mi-chemin des deux positions de références des parties données par le MOCAP. Et contrairement aux parties de base qui sont limitées en taille, les parties intermédiaires se propagent dans la totalité de l'espace restant.

La propagation, assimilable à une segmentation par croissance de régions, se fait en deux étapes. Les parties principales se propagent dans la zone de 10 cm. Ensuite, les parties intermédiaires se propagent pour combler l'espace restant. Une propagation, elle-même est décomposée en plusieurs petites itérations. Pendant une itération, chaque voxel déjà classé, se propage dans son voisinage, si celui-ci ne l'est pas encore. On limite le voisinage à six cases : Haut, bas, droite, gauche, devant et derrière.

Vu que les parties de base ont une propagation limitée par un rayon max, et vu que la propagation des parties complémentaires va de proche en proche, et ne saute pas les cases

vides, il se peut qu'à la fin, des parties soient non classées. Elles sont considérées comme pleines, mais restent dans la classe « inconnue ».

Après cette voxellisation, et connaissant la vérité terrain, nous allons discuter la caractérisation des paramètres libres sous jacents aux « random forests ».

3.5 Caractérisation des paramètres libres

Remarque

Ces évaluations portent sur le réglage ou « tuning » des paramètres. À notre connaissance, et notamment dans Shotton et al., ce « tuning » des paramètres est peu étudié et souvent fixé empiriquement. Une évaluation consiste à repasser par les étapes de voxellisation (si les paramètres évalués sont liés), d'apprentissage, de prédiction et de comparaison à la vérité terrain. Des tests de « cross validations » sont effectués avec les trois groupes constitués par **les postures utilisées qui sont au nombre de 1045 extraits de la base de données NSC13** (voir section 2.5). Nous avons utilisé la technique de cross-validation avec k échantillons « k-fold cross validation ». Nous avons fixé k à 3 i.e. la base est scindée en trois : un groupe d'échantillons des données « fold » parmi les 3 pour l'évaluation, les groupes restants servent à l'apprentissage. Puis nous faisons la moyenne des 3 tests pour afficher les résultats à comparer.

Nous exploitons un jeu de paramètres fixés empiriquement par défaut et listé dans le tableau 3-4. Les variations évaluées des paramètres seront mentionnées dans chaque cas. Le critère C_D a été fixé à une valeur très grande 500, pour laisser les autres critères d'arrêt définir la profondeur de chaque branche de l'arbre.

Nb postures pour les 3 groupes	1045
Nb total des voxels (***S***)	76 M
Nb arbres (***N***)	1
Nb Candidats par nœud (***d'***)	100
Seuil du gain d'info (***t***)	0.01
Min échantillons par nœud (***m***)	100
Ratio de sous-échantillonnage (α)	0.05
Min norme vecteur du descripteur (φ_{min})	0.02
Max norme vecteur du descripteur (φ_{max})	0.9
Moyenne norme vecteur du descripteur (φ_A)	0.46
Critère d'arrêt en fonction de la profondeur D (C_D)	500

Tableau 3-4 : Paramètres par défaut utilisés.

Remarque sur le temps de classification

Le temps de classification a été relevé pour chaque reconstruction car chaque posture induit un nombre variable de voxels et donc un coût CPU proportionnel durant leur labellisation.

Pour donner une référence comparative au lecteur, après avoir effectué l'expérimentation en utilisant toutes les postures, on présente toujours le temps de prédiction de la même posture (M6-369) qui est un espace voxellisé à 654 K voxels dont 82 K pleins et à classifier.

3.5.2 Nombre de Candidats par nœud

Le nombre de candidats (choisis aléatoirement) par nœud correspond au paramètre (**d'**) qui est détaillé dans le paragraphe ***notations*** de la section 3.3.4. Un candidat représente le vecteur du descripteur à tester pour décider du prochain nœud à suivre, gauche ou droit pour un résultat du test à 1 ou à 0 (voxel plein / voxel vide). La meilleure caractéristique vise à séparer au mieux les échantillons appris et donc choisir celle qui optimise le gain d'information associé lors de leur scission en deux classes. Les questions qui se posent sont : si l'on évalue plus de candidats, est-ce que l'on obtiendrait toujours une meilleure sélectivité au niveau du nœud ? Est-ce que cette meilleure sélectivité au niveau d'un nœud unique aurait un bon impact sur la performance globale de l'arbre construit ? Le tableau 3-5 répond à ces questions.

Nb Candidats	Nb Nœuds	Temps d'apprentissage	Temps de classification ms	Profondeur médiane	Profondeur max	Entropie des feuilles	Gain d'info des nœuds	Ratio de classification des feuilles
100	94461	1mn 17s	77	21	116	0,927	0,163	0,778
200	91507	2mn 7s	75	22	95	0,892	0,165	0,799
300	90575	2mn 50s	75	21	95	0,874	0,166	0,808
500	88559	4mn 27s	76	21	112	0,903	0,175	0,807
700	87395	6mn 38s	73	21	96	0,891	0,175	0,814
1000	85983	9mn 47s	76	21	109	0,88	0,174	0,821
1500	83163	14mn 4s	77	21	102	0,871	0,176	0,831
2000	83091	18mn 2s	75	21	104	0,883	0,18	0,829
3000	82831	28mn	74	21	96	0,89	0,183	0,829
5000	80295	54mn	75	21	95	0,883	0,188	0,836

Tableau 3-5 : Résultats et paramètres d'arbre d'apprentissage.

Ainsi, pour un faible nombre de candidats, on a besoin de plus de nœuds pour satisfaire les contraintes d'arrêt de croissance de l'arbre. Le temps d'apprentissage est proportionnel au nombre de candidats qui offre le compromis entre la qualité et ce coût CPU . Par contre, pour le temps de calcul de la classification en ligne, ce paramètre n'a pas d'impact. L'entropie des feuilles a des variations similaires au ratio de classification des feuilles. Le ratio de classification des feuilles est le ratio des voxels correctement prédits par la feuille. Ces deux paramètres incluent un calcul qui a été effectué sur tous les nœuds ayant mené à cette feuille en question, tandis que le gain d'info des nœuds considère le seul nœud en question.

Pour conclure, plus on considère de candidats, plus les nœuds ont une meilleure sélectivité (gain d'info). Par contre, le pouvoir discriminant des feuilles générées n'augmente pas de la même manière et commence même à régresser.

Ratio mAP : « mean Average Precision »

Le terme mAP « mean Average Precision », initié par Shotton et al., permet de quantifier la précision de reconstruction. Le mAP représente le ratio des centres prédits avec une erreur en dessous d'un certain seuil. Nous avons fixé ce seuil à 0.1 m.

Le mAP est privilégié et non la moyenne sur la précision des centres reconstruits car toutes les valeurs ne sont pas toujours disponibles.

L'influence du nombre de candidats est évaluée ci-après sur ces critères mAP et le taux de classification (figure 3-15). Ils représentent une moyenne des trois tests de cross-validation.

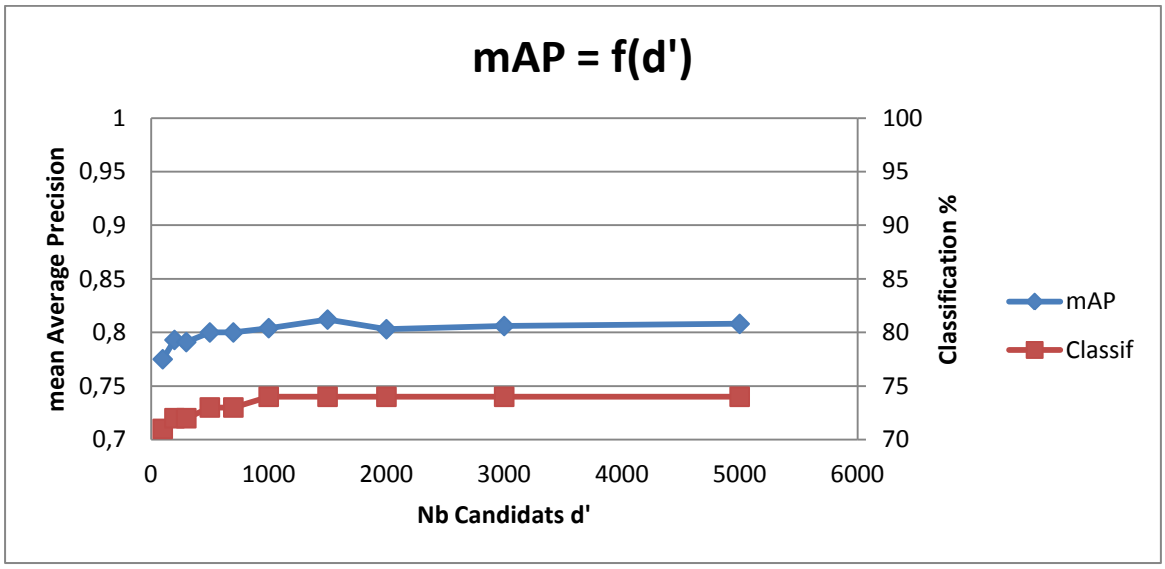

Figure 3-15 : mAP en fonction du nombre de candidats d'.

La figure 3-15 montre la variation du mAP et le taux de classification des voxels en fonction du nombre de candidats. On constate une hausse du mAP jusqu'à l'intervalle [1000,2000] puis une légère baisse non significative. Par contre, le temps d'apprentissage d'un seul arbre pour les trois tests, a un temps de calcul qui passe de quelques minutes à une heure, il faut donc choisir le meilleur compromis selon nos critères.

Ces évaluations nous amènent à privilégier une valeur de nombre de candidats autour de 1000.

3.5.3 Le sous-échantillonnage de la base des voxels à apprendre

La base de données contient des postures semblables, et une même posture contient des voxels dont le voisinage est semblable, ceci crée des données redondantes impactant l'apprentissage. Le rapport de sous-échantillonnage (α) consiste à alléger le nombre d'échantillons à apprendre. Quand on fixe ce rapport, il est considéré de façon indicative. On fixe le nombre de sous-échantillons à atteindre. Ensuite, pour chaque classe ou partie du corps, on cible la proportion qui garantit l'égalité de son nombre d'échantillons avec le reste des classes. Si la classe en

question n'a pas assez d'échantillons pour atteindre ce nombre, alors la totalité des échantillons de cette classe est utilisée.

La figure 3-16 montre l'augmentation du mAP en fonction du rapport de sous échantillonnage. Par contre, à partir de la valeur 0.2, la classification diminue alors que le mAP augmente. Ceci illustre la non-corrélation de la classification avec la précision de l'estimation des centres des parties corporelles. Cette non-corrélation n'a été observée que pour ce paramètre (rapport de sous-échantillonnage) car il est directement responsable de sur-apprentissage (« over fitting »).

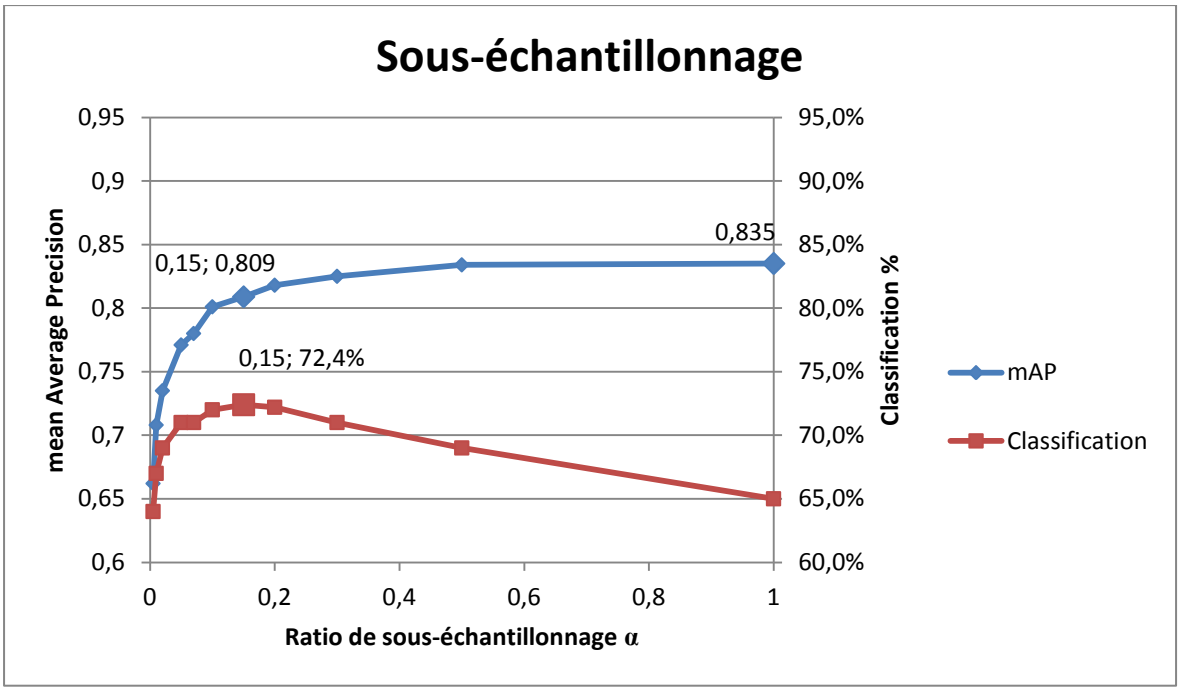

Figure 3-16 : mAP et Classification vs. sous-échantillonnage.

Sous échantillonnage	Nb échantillons	Temps d'apprentissage	Nb Nœuds	Temps de classification ms	Profondeur médiane	Profondeur max	Entropie des feuilles	Gain d'info des nœuds	Ratio de classification des feuilles
0,005	241 K	21 s	10341	55	16	83	1,36	0,18	0,641
0,01	488 K	28 s	20499	60	17	90	1,22	0,176	0,684
0,02	977 K	41 s	39945	67	18	83	1,08	0,168	0,724
0,05	2,4 M	1 mn 16 s	93355	77	20	98	0,939	0,165	0,776
0,07	3,4 M	1 mn 40 s	126833	83	21	104	0,894	0,162	0,792
0,1	4,8 M	2 mn 11 s	174055	93	22	105	0,826	0,158	0,814
0,15	7,0 M	3 mn 12 s	245245	100	22	117	0,776	0,155	0,832
0,2	9,0 M	3 mn 52 s	300699	109	23	115	0,731	0,149	0,848
0,3	12,3 M	4 mn 53 s	399213	118	23	113	0,71	0,149	0,855
0,5	16,9 M	6 mn 33 s	530459	136	24	124	0,671	0,143	0,866
1	24,7 M	8 mn 56 s	738773	159	24	123	0,648	0,14	0,876

Tableau 3-6 : Paramètres de l'arbre en fonction du sous-échantillonnage.

Le tableau 3-6 indique les paramètres de l'arbre de décision en fonction de ce paramètre. Le nombre de nœuds croît logiquement avec le nombre d'échantillons à apprendre ainsi que le coût CPU.

Si le temps de classification est important pour avoir un temps de réponse réduit et si l'application n'a pas besoin de la posture avec précision, il est possible de choisir un compromis entre la précision et le temps de classification. Une étude similaire pourrait être menée sur l'influence de ce paramètre durant le processus de reconstruction en ligne.

3.5.4 Paramètres intrinsèques à l'arbre de décision

Ces évaluations caractérisent l'influence du paramètre (C_D). Rappelons que ce critère d'arrêt influence directement la profondeur D_{MAX} de l'arbre ; les nœuds sont transformés en feuilles lorsque le critère est vérifié.

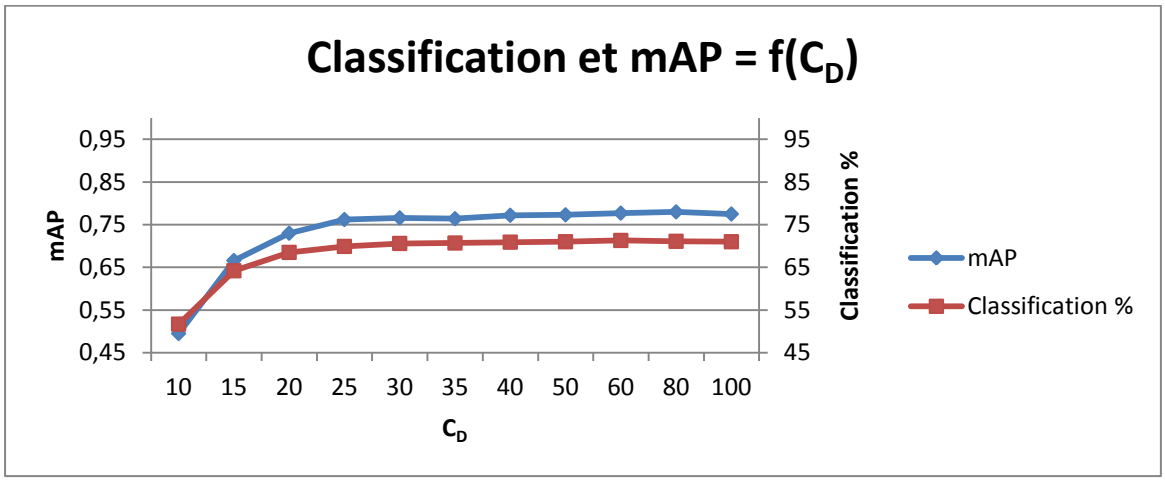

Figure 3-17 : Classification et mAP vs. profondeur maximale.

La figure 3-17 montre que l'absence des nœuds, au-delà des profondeurs 25, impacte très peu les performances de classification et l'estimation des centres des parties corporelles. Même pour un arbre de profondeur supérieure à 100, sa partie utile n'est contenue que dans les profondeurs inférieures à 25 ; le reste ne concerne que des petites branches non significatives.

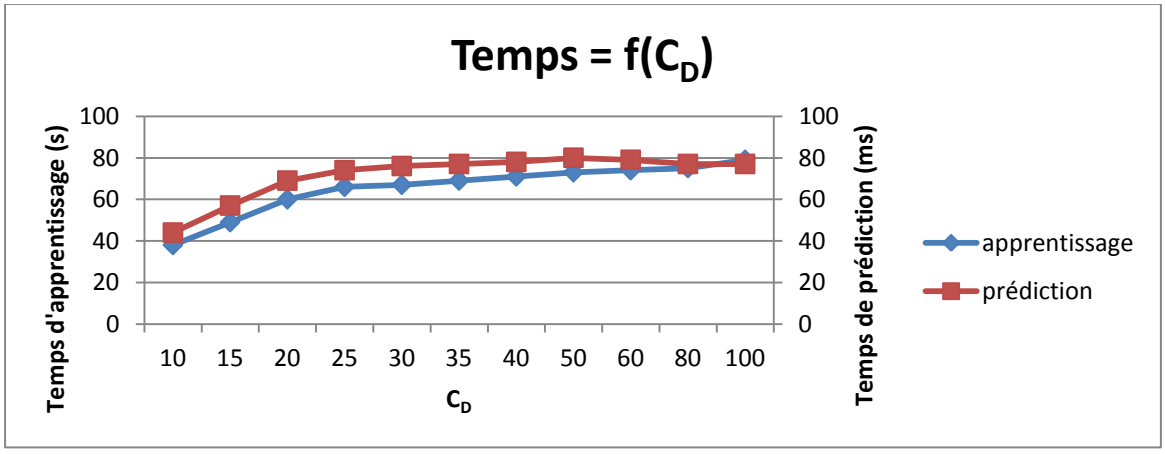

Figure 3-18 : Temps d'apprentissage et temps de classification vs. profondeur maximale.

La figure 3-18 illustre logiquement la corrélation entre temps d'apprentissage et temps de classification. Le nombre de nœuds est à peu près proportionnel au temps d'apprentissage. Pour les profondeurs 10 à 25, le nombre de nœuds croît et a une influence sur le coût CPU « online » de classification.

La figure 3-17 et la figure 3-18 permettent de conclure que même si les profondeurs au-delà de 25 n'ont pas un impact sur le résultat, leur présence ne perturbe ni le temps d'apprentissage, ni le temps d'exécution qui restent pratiquement inchangés. Cela ne gène donc pas de garder la profondeur totale de l'arbre. Par contre, le nombre 25 est propre à notre application. Dans cette expérimentation 617 344 échantillons ont atteint la profondeur 25 sur 2 453 064 en totalité, d'où un pourcentage de 25%. On peut donc conclure, au moins dans cette expérimentation, que ce ne sont que 25 % des données qui contribuent réellement aux performances de la classification.

3.5.5 Paramètre de seuil du gain d'information

Le gain d'information par le seuil (***t***) dans chaque nœud impacte aussi la génération de l'arbre. Pour rappel, lors de l'apprentissage, ce seuil est testé pour décider si le nœud doit se diviser encore ou se transformer en feuille.

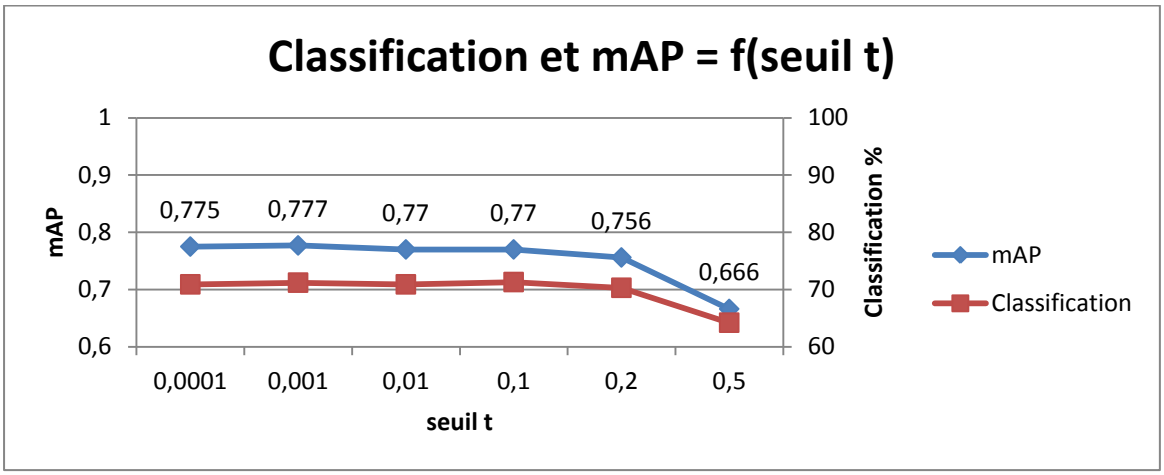

Figure 3-19 : Classification et mAP vs. seuil du gain d'information.

La figure 3-19 montre que le gain d'information dégrade les performances pour des valeurs supérieures au seuil 0,2.

Au-delà des performances, évaluons l'impact du gain d'information sur la structure de l'arbre.

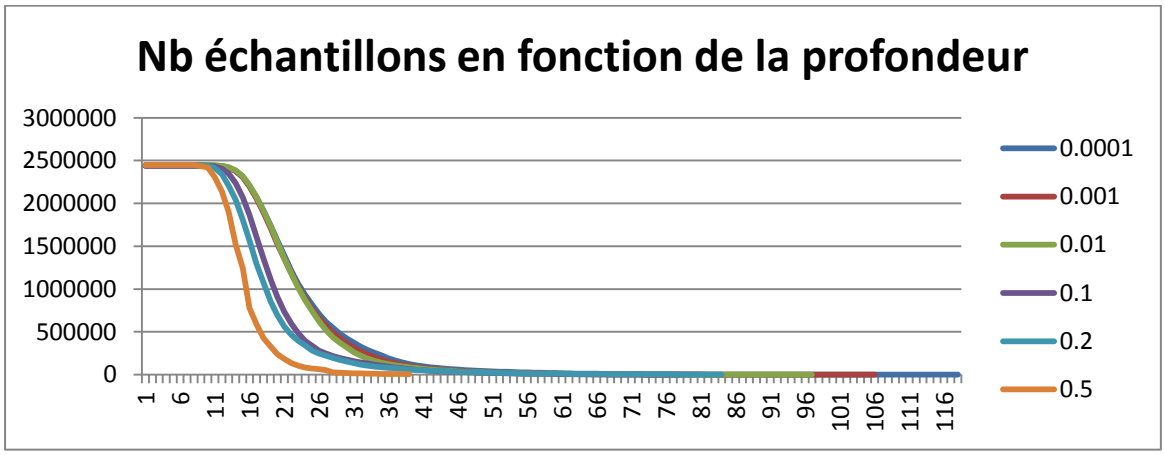

Figure 3-20 : Nombre d'échantillons vs. profondeur pour différentes valeurs du seuil de gain d'information.

On rappelle qu'un échantillon correspond à un voxel. Il est injecté à l'entrée de l'arbre pour le processus d'apprentissage. La figure 3-20 nous permet de remarquer que pour le seuil 0.1 du gain d'information, il y a une large baisse du nombre d'échantillons atteignant les profondeurs 20 à 30 et au delà. La figure 3-21 illustre l'influence du seuil sur le nombre de noeuds. Le nombre total de nœuds passe de 99 000 à 48 000, et ceci sans impact sur les performance de classification et précision (mAP) tel que le montre la figure 3-19. Au final, ces évaluations nous amènent à fixer ce seuil à 0.01. pour ne pas avoir à sur-dimensionner l'arbre ni le charger en quantité d'informations inutiles.

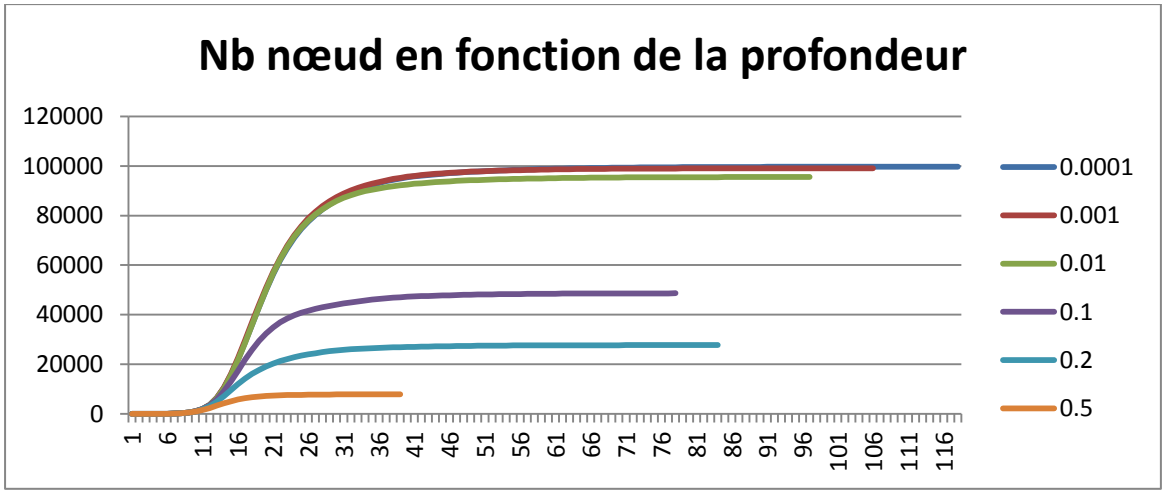

Figure 3-21 : Nombre total de nœuds vs. profondeur pour différentes valeurs du seuil de gain d'information.

3.5.6 Fenêtre des vecteurs de descripteurs

Chaque descripteur (v) de l'arbre de décision consiste en un vecteur sélectionné parmi un ensemble de candidats (ϕ). Cet ensemble de descripteurs candidats (ϕ) est le même quelque soit sa profondeur. Celles-ci sont échantillonnées dans un volume tel que :

- La moyenne des normes des vecteurs est égale à (ϕ_A = 0.3 m)
- Min et max des normes des vecteurs sont fixés à (ϕ_{min} = 0.02 m, ϕ_{max} : variable)

Les évaluations préliminaires sur un ϕ_A variable n'ont pas eu d'impact sur le système. Ceci peut s'expliquer par le grand nombre de descripteurs générées pour (ϕ).

Les expérimentations visant à faire varier (ϕ_{max}) ont montré un impact sur l'apprentissage et les taux de classification. L'influence de ce paramètre se justifie aussi par son impact indirect sur l'apprentissage, car il influence la localité du descripteur. Un descripteur trop local ne couvre que la zone de la partie corporelle à apprendre ; il est alors moins discriminant pour classer le voxel. En contrepartie, un descripteur trop grand devient très sensible aux variabilités de postures. La figure 3-22 affiche les critères mAP taux de classification pour différentes tailles de voisinage/fenêtre. La stratégie d'échantillonnage utilisée est celle UniNorme. Les résultats sont satisfaisants pour une fenêtre de taille supérieure à 0.4 m. Pour une fenêtre de taille supérieure à 0.7 m, le taux de classification ainsi que la précision moyenne décroissent sensiblement. Plus ce voisinage est grand, plus le risque de sur-apprentissage augmente. Ces investigations nous invitent à privilégier une taille comprise entre les valeurs précitées.

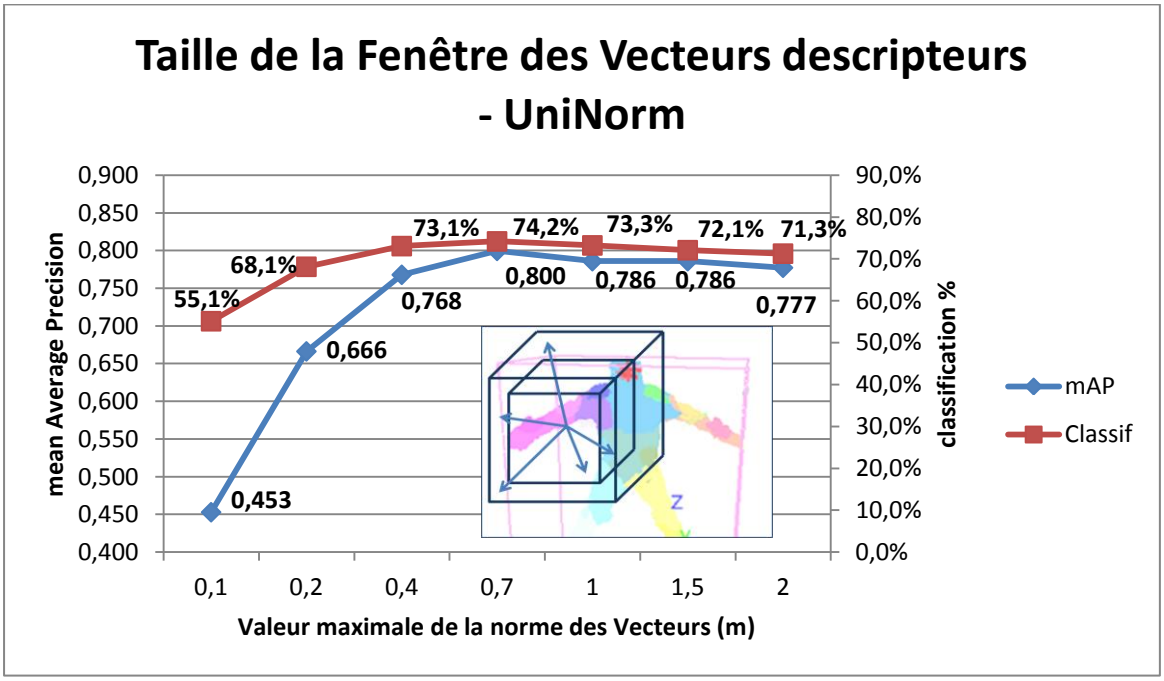

Figure 3-22 : Taille de la fenêtre des Vecteurs de descripteurs – UniNorme.

3.5.7 Nombre d'arbres dans la forêt

Le nombre d'arbres noté N est important, car il est proportionnel au coût CPU « online »

La figure 3-23 indique la précision mAP et le taux de classification pour des apprentissages effectués avec un nombre d'arbres allant de 1 à 20 (pas variable). Les performances croissent avec le nombre d'arbres. Pour la précision, on gagne 5 % en passant de 1 à 2, et 14 % de 1 à 20. Ce gain de performances est à mettre en regard des contraintes CPU i.e. le temps d'apprentissage et le temps de classification qui sont détaillés dans le tableau 3-7.

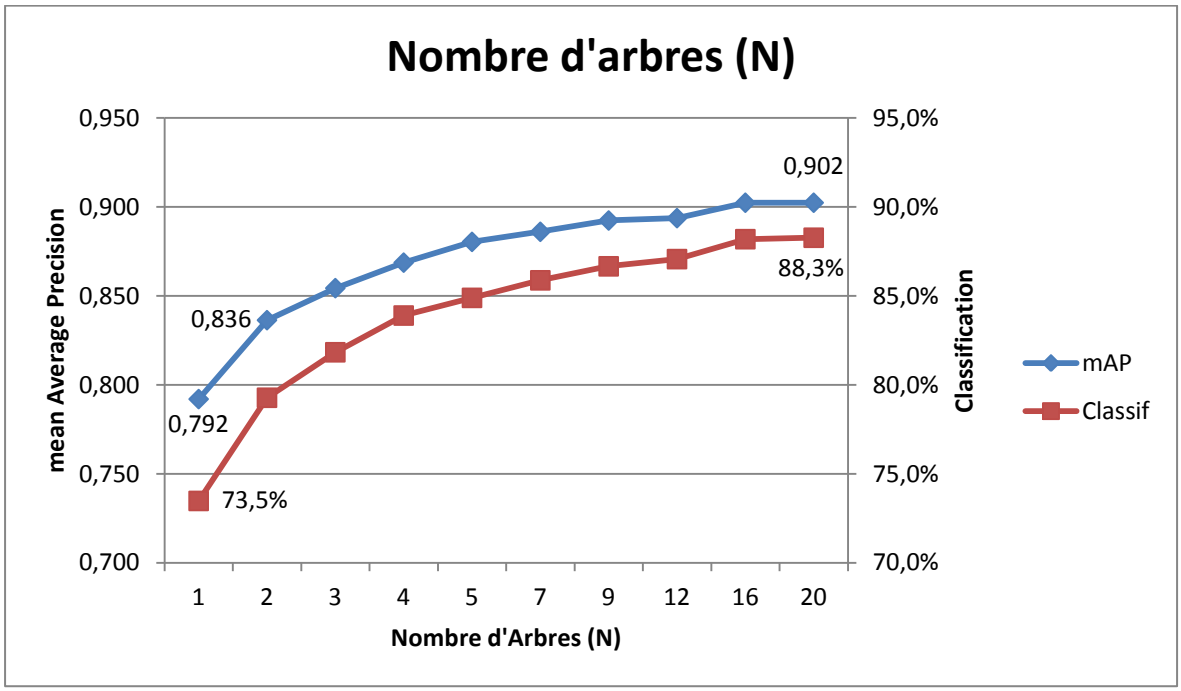

Figure 3-23 : Nombre d'arbres dans la forêt – UniNorm.

Nb Arbres	Temps d'apprentissage	Temps de classification
1	9 mn 44 s	077 ms
2	18 mn 40 s	136 ms
3	27 mn 42 s	210 ms
4	37 mn 57 s	300 ms
5	47 mn 5s	370 ms
7	1 h 8 mn	546 ms
9	1 h 24 mn	724 ms
12	1 h 51 mn	1 s 037 ms
16	2 h 30 mn	1 s 566 ms
20	3 h 7 mn	2 s 149 ms

Tableau 3-7 : Temps d'apprentissage et de classification vs. nombre d'arbres.

Le tableau 3-7 énumère les temps d'apprentissage et de classification en fonction du nombre d'arbres. Ces temps sont fonction de la machine utilisée. Le calcul de la classification a été effectué avec un seul « thread » et vu l'indépendance du calcul de la classification avec chaque arbre, il est divisible par le nombre de « thread » dans la limite d'un « thread » par arbre. Le temps incompressible est donc de 77 ms, et les ressources nécessaires (nombre de « thread ») dépendent du nombre d'arbres souhaité.

Ici, le temps d'apprentissage est assez réduit et ne représente pas le goulet d'étranglement du processus. C'est grâce à l'optimisation de la structuration des données (un accès mémoire pour évaluer une caractéristique) et à notre transformation des données en un volume dans un espace cartésien (plus besoin de divisions pour rectifier la distance dans une image de profondeur).

La figure 3-24 illustre qualitativement la classification selon le nombre d'arbres utilisés. Pour 20 arbres, le résultat est proche de l'image vérité terrain. Les gains sont surtout observés lors du passage du nombre d'arbres de 1 à 3.

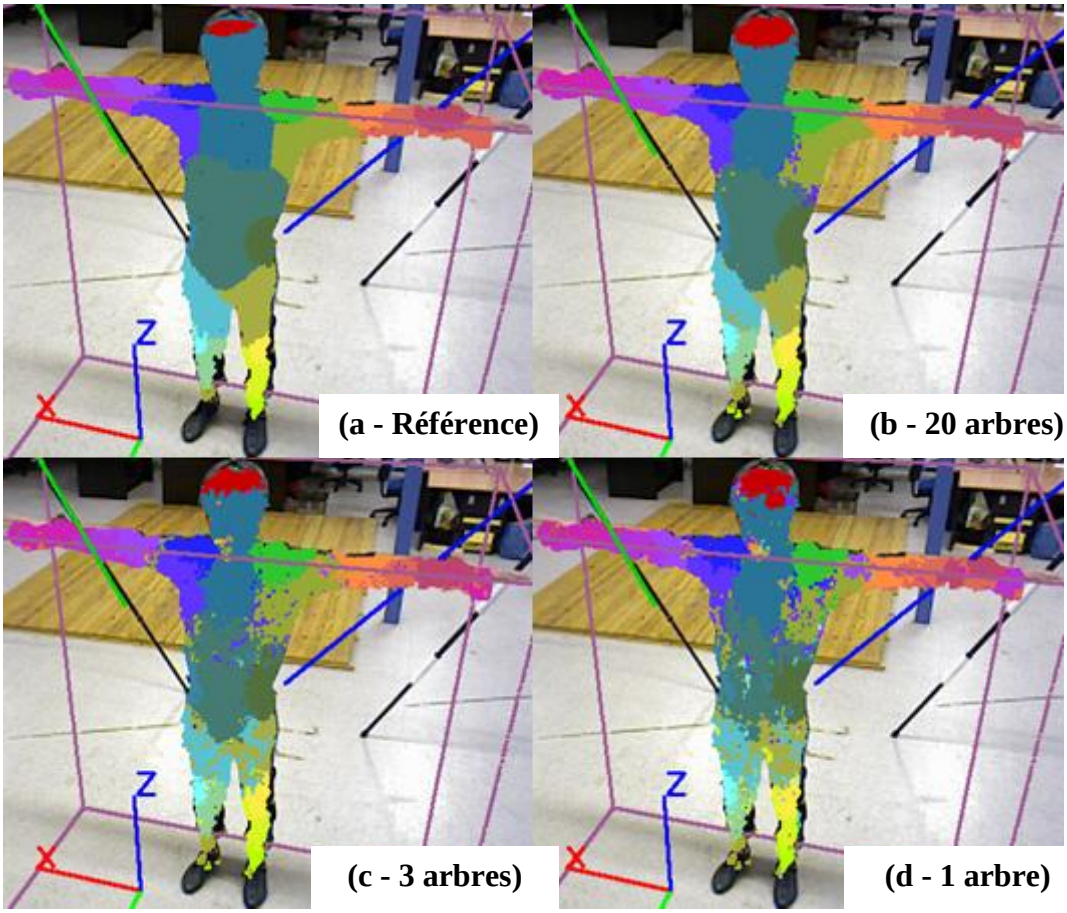

Figure 3-24 : Exemples de classification selon le nombre d'arbres.

3.5.8 Vote vs. pondération

Rappelons que la classification repose sur un mécanisme de vote ou de pondération énoncé en section 3.3.4 et met en jeu la forêt d'arbres.

La pondération « additionne » les histogrammes des feuilles atteintes dans les différents arbres alors que le vote permet à chaque arbre de choisir une classe pour ensuite procéder à un vote parmi ces classes obtenues.

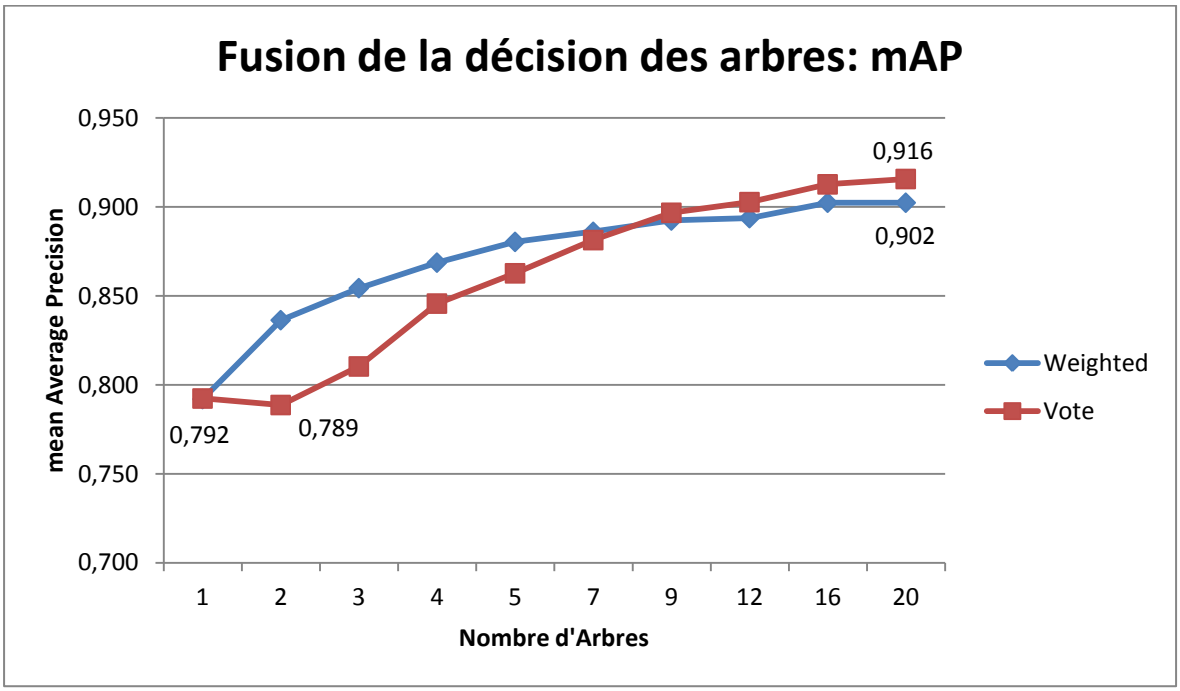

Figure 3-25 : Comparaison des techniques de Vote selon le mAP.

La figure 3-25 montre que la précision moyenne n'est meilleure en utilisant la technique du vote que pour un nombre d'arbres supérieur à 7.

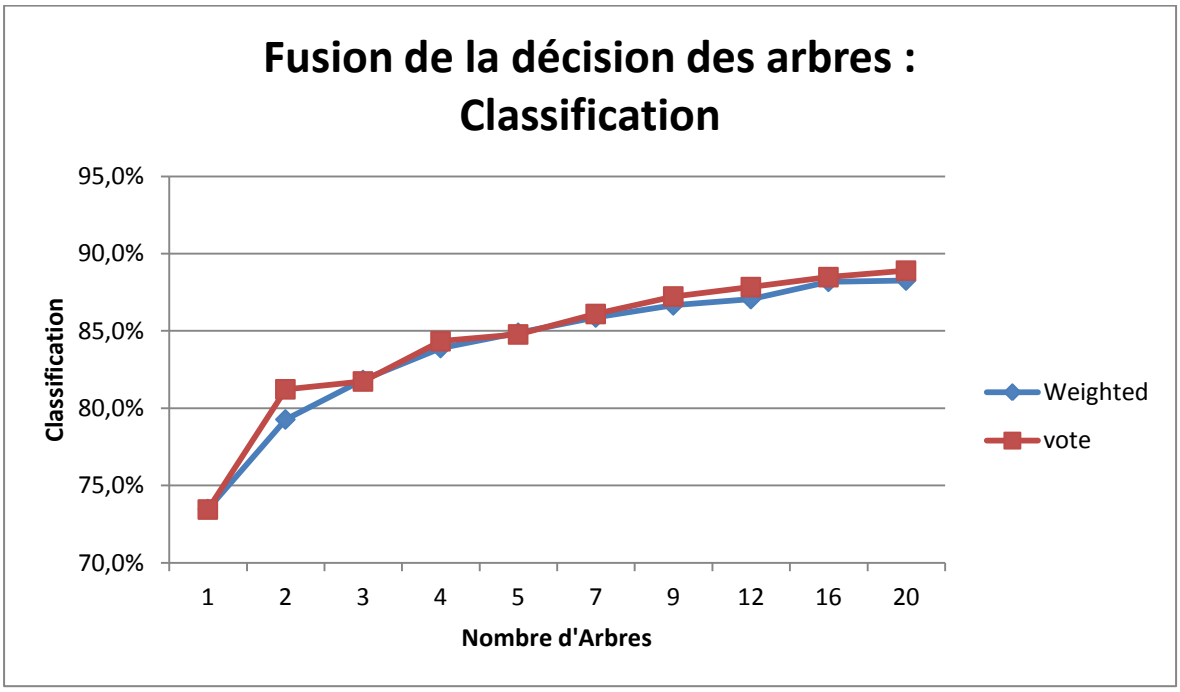

Figure 3-26 : Comparaison des techniques de vote selon la classification.

A contrario, la figure 3-26 montre que la stratégie n'a pas de réel impact sur la classification. Pour le choix de la stratégie, il faut donc plutôt se fier au paramètre mAP.

3.5.9 Récapitulatif des paramètres libres

Notre étude sur les paramètres libres nous a permis de caractériser (et donc de « tuner ») l'influence de ceux-ci sur les performances du système et sur le coût CPU « online » (classification) et « offline » (apprentissage).

Paramètre	Dénominations	Valeur optimale	Justification
d'	Nombre de Candidats par nœud	1000	Ralenti proportionnellement le temps d'apprentissage, mais n'impacte pas le temps de prédiction.
α	Facteur de sous échantillonnage de ***S***	0,3 - 0,5	Ralenti proportionnellement le temps d'apprentissage, impacte le temps de prédiction mais pour un meilleur résultat (compromis qualité/performance).
D	Profondeur de l'arbre	>100 ou Non limitée	La limitation par la profondeur est une mauvaise approche, car elle crée un déséquilibre dans l'entropie des feuilles.
t	Seuil posé sur le critère du gain d'information	0,01	Les valeurs au dessus de 0,1 impactent l'arbre, et les valeurs en dessous de 0,01 créent des branches très profondes sans gain de qualité.
φ φ_{max}	Ensemble des caractéristiques candidates	φ_{max}=0,7 m UniNorm	La valeur maximale supérieure à 0,7 m ne présente pas davantage et réduit la localité du descripteur. La méthode UniNorm réparti les normes des vecteurs.
N	Nombre d'arbres de décision constituant la forêt de décision	1 – 3	Le temps d'apprentissage est proportionnel (même si parallélisable, il faudrait plus de ressources), pareil pour le temps de prédiction (compromis qualité/performance).
m	Seuil posé sur le critère du nombre minimal d'échantillons par nœud	100	En dessous de cette valeur, il n'est pas pertinent de construire un histogramme représentatif. Au-dessus, on limite la croissance de l'arbre.

Tableau 3-8 : Réglage (« tuning ») des divers paramètres libres.

Le tableau 3-8 récapitule les compromis observés lors des réglages . Ces observations permettent de fixer les valeurs qui nous semblent opportunes et offrant un compromis entre performances et considérations CPU. Ce jeu de paramètres sera privilégié pour nos évaluations dans le chapitre suivant.

Toute comparaison avec les travaux de Shotton et al. est ici délicate, car le descripteur utilisé est différent. Néanmoins et à titre indicatif, nous avons fixé d' = 1 000 alors que Shotton et al. suggère d'=100 000. Shotton et al. privilégient N=3 arbres avec profondeur D=20. Les autres paramètres libres restent peu explicités et discutés dans Shotton et al.

3.6 Conclusion

Ce chapitre formalise notre approche, présente les « random forests » (outil essentiel de notre classifieur), enfin exhibe les paramètres libres associés. Nous avons alors proposé de

nombreuses évaluations pour caractériser les performances de notre approche et régler (« tuner ») les paramètres libres sous-jacents. Deux contributions, au cœur des travaux, nous semblent à ressortir de ce chapitre.

Tout d'abord, nous avons proposé une approche multi capteurs kinect qui repose sur un descripteur 3D après voxellisation de l'espace reconstruit par triangulation des capteurs. Même si nous exploitons les « random forest », notre approche se démarque ici de celle de Shotton et al. qui est mono-capteur et repose sur des descripteurs inférés dans l'image de disparité.

Nous avons implémenté notre propre librairie de forêt de décision pour maîtriser l'outil et ainsi cerner les paramètres libres associés.

Deuxièmement, nous avons montré que le réglage ou « tuning » des paramètres libres du système complet (voxellisation, apprentissage, etc.) influencent grandement les performances de classification, précision, et coût CPU. Cette étape est essentielle alors que ce « tuning » est peu étudié dans la littérature. Le chapitre suivant, fort du tuning effectué ici, se focalise logiquement sur des évaluations qualitatives/quantitatives de notre approche et une comparaison avec l'approche de Shotton et al.

Chapitre 4 Évaluations et étude comparative

Ce chapitre présente et discute les évaluations de notre approche et sa comparaison avec l'approche mono kinect de Shotton et al. Les bases de données NSC13 et IRSS35 sont respectivement exploitées pour les évaluations propres et la comparaison. La seconde, avec ses 35 marqueurs, permet d'extraire un squelette comparable à Shotton et al. et donc une comparaison. Pour rappel, ces bases sont explicitées dans la section 2.5.

4.1 Évaluations

Après avoir étudié l'influence des paramètres libres dans le chapitre précédent, nous allons maintenant présenter des évaluations qualitatives et quantitatives avec les paramètres ainsi « tunés » et ainsi caractériser les performances de notre approche.

4.1.1 Base de données et paramètres libres associés

Le tableau 4-1 présente le jeu de données relatives à l'exploitation de la base NSC13. La segmentation utilisée pour extraire la silhouette de la personne est parfois inexploitable pour la reconstruction. Quelques heuristiques permettent de filtrer des segmentations a priori erronées. Mais quand une posture a plus que deux parties du corps manquantes, elle n'est pas utilisée dans l'apprentissage. Une partie du corps est considérée comme manquante si elle comporte moins que 100 voxels reconstruits. Le nombre de parties corporelles reconstruites est supposé de 25 ; ce nombre incluant les parties du corps principales et les parties du corps intermédiaires (Cf. section 3.4.2).

Nombre de séquences	**5**
Nombre total d'images	**1 951**
Nombre de postures utilisées	**1 045**
Nombre total de voxels |S|	**76 M**
Nombre de classes	**25**

Tableau 4-1 : Base de données utilisée.

La répartition des échantillons en classes est indiquée par la figure 4-1. Le nombre de voxels par classe est évidemment très déséquilibré comme illustré sur l'exemple de posture M2-99 représenté sur la figure 4-2.

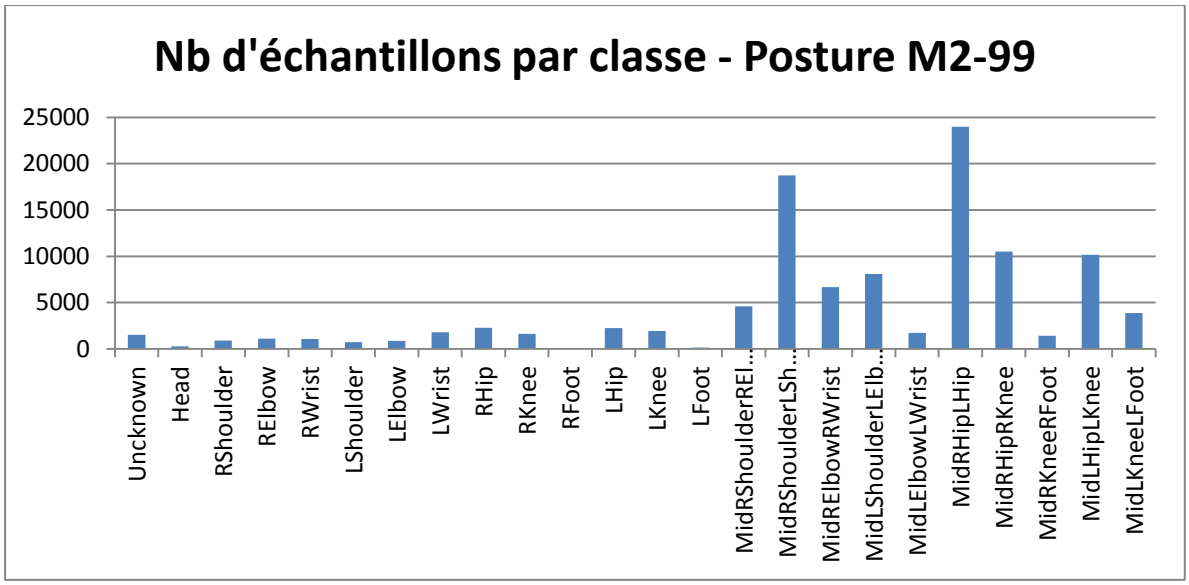

Figure 4-1 : Nombre de voxels par partie corporelle pour la posture M2-99.

La figure 4-2 illustre le cube englobant le volume de la personne et les différentes parties du corps représentées par des couleurs différentes.

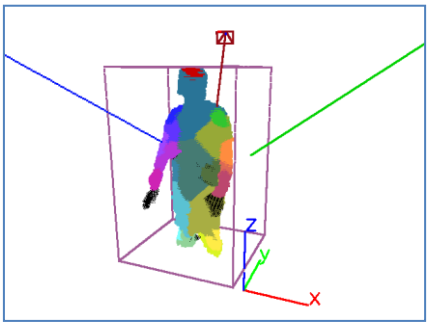

Figure 4-2 : Volume de travail pour la posture M2-99.

Un voxel sera dénommé « échantillon » de la base de données. Le déséquilibre du nombre d'échantillons par classe influe sur les performances du classifieur. Un sous-échantillonnage préalable est donc réalisé pour rééquilibrer le nombre d'échantillons par classe. Ainsi, pour P classes, on limite les échantillons de chaque classe à un ratio de 1/P. Si le ratio d'une classe est initialement inférieur à 1/P, alors tous les échantillons de cette classe sont gardés.

Les évaluations menées ici sont obtenues avec le « tuning » suivant des paramètres libres introduits au chapitre 3 :

Nombre de Candidats par nœud (d')	1000
Ratio de sous-échantillonnage (α)	0.5
Profondeur max (D)	500
Limite gain d'information (t)	0.01
min norme vecteur descripteur (ϕ_{min})	0.02
Max norme vecteur descripteur (ϕ_{max})	0.9
Nombre d'arbres (N)	1
Min échantillons par nœud (m)	100

Tableau 4-2 : Valeurs des paramètres libres.

Les évaluations ci-après s'appuient sur un apprentissage effectué en validation croisée i.e. apprentissage effectué sur deux tiers de la base et la classification sur le tiers restant de la base.

4.1.2 Observations sur le processus d'apprentissage

Le tableau 4-3 présente un exemple d'arbre appris avec deux tiers de la base de données NS-Cap13.

Sous-échantillons appris	25 M
Profondeur max atteinte	108
Entropie moyenne	0.732
Nb total des nœuds	704 327
Nb feuilles	352 164
Gain d'info moyen	0.178
Temps d'apprentissage	1h 37 mn

Tableau 4-3 : Caractérisation des « random forest » appris.

La figure 4-3 illustre la génération de l'arbre sachant qu'une étape/itération est une subdivision d'un nœud parent en deux nœuds fils (axe des abscisses). Le trait (bleu) représente le nombre d'échantillons nécessaire au calcul lors des itérations de construction de l'arbre. L'histogramme (rouge) représente le temps nécessaire au calcul de chaque itération.

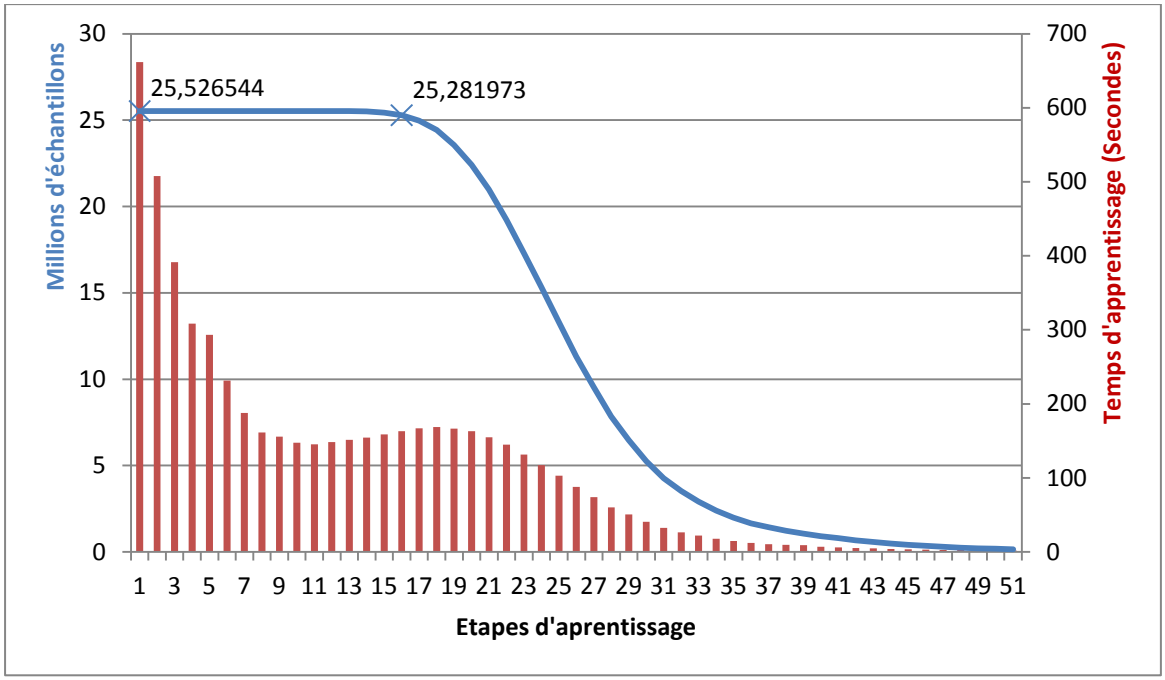

Figure 4-3 : Nombre d'échantillons des parties corporelles à apprendre et temps d'apprentissage en fonction des étapes d'apprentissage.

Cette figure amène quelques commentaires. L'étude ne porte pas sur le nombre initial d'échantillons $|S_0|$ à l'entrée de l'apprentissage i.e. $|S_0|$ est fixé a priori (25 M), mais sur la variation/diminution du nombre d'échantillons considérés durant l'apprentissage.

Rappelons que l'apprentissage commence par la racine (premier nœud) de l'arbre dans lequel on va injecter toute la base de données d'échantillons S_0. Ce nœud est affiché à la première position dans l'axe des abscisses. L'ordonnée de la courbe bleue en ce point correspond à la valeur de la totalité de la base (25 M). Lorsqu'un nœud se divise en deux nœuds fils, la base de données est, elle aussi, divisée en deux en fonction du résultat du test du nœud. Mais la quantité affichée par la courbe bleue reste inchangée au début, car elle affiche la somme des échantillons utilisés par la totalité des nœuds dans une profondeur donnée. Cette quantité décroît lorsqu'un nœud vérifie un critère d'arrêt ; il se transforme alors en feuille. Les données associées servent alors à construire l'histogramme contenu dans cette feuille et ne seront plus considérées dans la suite de la génération de l'arbre.

On réfère par profondeur utile, la profondeur ayant une influence sur le traitement des données. Ainsi, une troncature de l'arbre à une profondeur de 50 et non de 51 ne néglige qu'une quantité infinitésimale des données. Au-delà de la valeur 51, 0.5 % des données sont négligées tandis que pour la valeur 16, on garde plus que 98 % des données.

Ces observations nous aident à caractériser l'arbre idéal pour lequel tous les nœuds se transforment en feuilles à la même profondeur et le coût CPU (nombre de tests) est indépendant du chemin parcouru dans l'arbre. En pratique, les critères d'arrêts induisent des chemins ayant des profondeurs très différentes et donc des coûts CPU variables. C'est pour cela que l'on peut conclure qu'en pratique, il est possible de tronquer l'arbre entre les profondeurs 16 et 50 e.g. au niveau 35.

La figure 4-3 illustre aussi le temps d'apprentissage de chaque étape. Celui-ci notifié en rouge sur la figure est le temps nécessaire au $i^{ème}$ nœud pour évaluer |Si| échantillons dans ***d'*** tests des nœuds candidats (ϕ). Cela permet de conclure sur le meilleur candidat à retenir. À la première itération, ce sont $|S_0|$échantillons qui parcourent ***d'*** candidats. À la deuxième itération, ce sont $|S_1|$ échantillons qui parcourent ***d'*** candidats d'un côté et $|S_2|$ échantillons qui parcourent ***d'*** candidats de l'autre. Entre la première et la deuxième itération, il y a donc le même nombre de parcours de candidats par des échantillons. Dans notre implémentation, on utilise le « Multithreading ». C'est-à-dire que chaque « thread » (ou partie du programme) qui a ses propres données, peut fonctionner en parallèle sur une machine multi-cœurs. Au niveau S_0, l'exécution utilise les mêmes données (mêmes échantillons et mêmes candidats à mettre à jour), l'exécution se fait donc avec un unique « thread ». À partir du premier niveau de profondeur, notre implémentation répartit automatiquement le traitement en « threads » séparés. Le traitement de S_1 et de S_2 se calcule donc en parallèle. Le temps de la première étape est de 662 secondes. Théoriquement, la deuxième itération devrait se dérouler dans la moitié du temps, mais par contre, elle dure 508 secondes. Effectivement avec deux « threads », l'efficacité de la parallélisation est de :

$$Efficacité\ de\ la\ parallélisation\ \text{à}\ l'étape\ 1\left(\frac{662}{2}\Big/_{508}\right) = 65\%$$

À chaque étape, les données sont réparties sur les différents nœuds. On lance autant de « threads » que de nœuds. Pour une optimalité de temps d'exécution, le nombre de « threads » est limité à 16 même si on dispose de plus de 16 nœuds. À partir de là, au niveau 4, les nœuds sont traités à tour de rôle, cela induit aussi une nouvelle perte d'efficacité. Cette nouvelle perte d'efficacité du nombre de nœuds à traiter supérieur au nombre de « threads » se traduit par la hausse du temps d'itération observé entre la profondeur 11 et 19.

À partir du niveau 19, des nœuds se transforment en feuilles et le nombre d'échantillons total à traiter diminue (observé par la courbe bleue) cela entraîne une baisse proportionnelle du temps d'apprentissage par étape.

4.1.3 Évaluations qualitatives

La figure 4-4-(a) illustre une image RGB issue du Kinect et une superposition du squelette obtenu par les marqueurs du système de MOCAP. La figure 4-4-(d) représente le canal « Depth » du Kinect. Les figure 4-4-(b) et (c) représentent la vérité-terrain en termes de segmentation, labellisation des parties corporelles. Les disques colorés représentent les centres des principales parties du corps correspondantes. Les figure 4-4-(e) et (f) représentent la classification des parties corporelles et leur localisation après application du « mean shift ». Les centres associés sont connectés pour reconstituer le squelette. La reconstruction obtenue, malgré une labellisation imparfaite, est proche de la vérité-terrain fournie par le MOCAP.

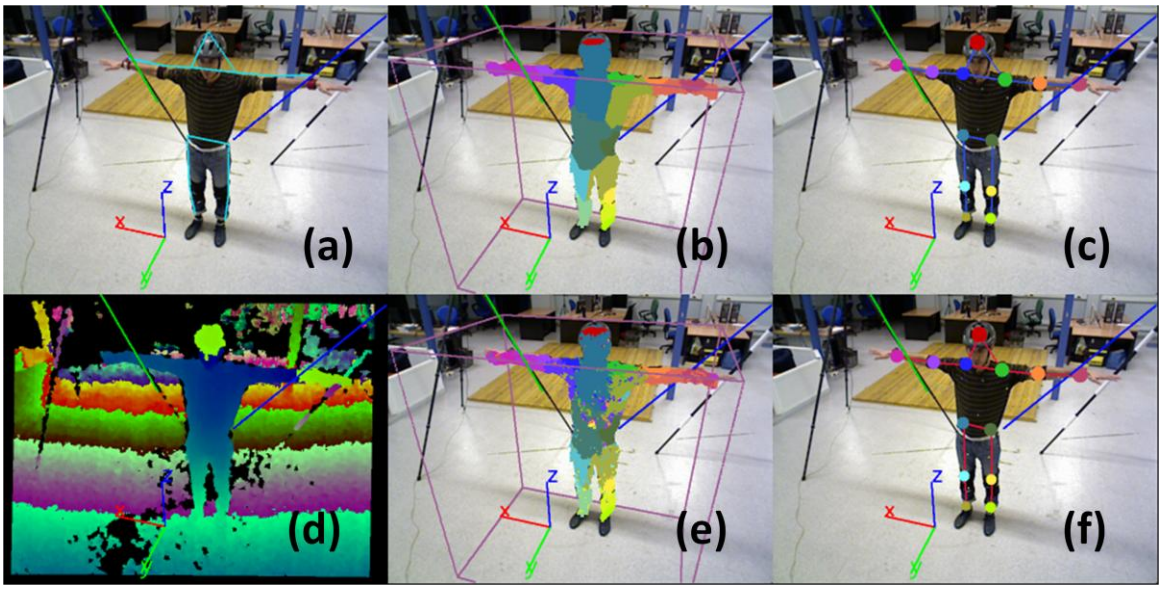

Figure 4-4 : Vérité terrain avec labellisation des parties corporelles et centres des joints – M3-236.

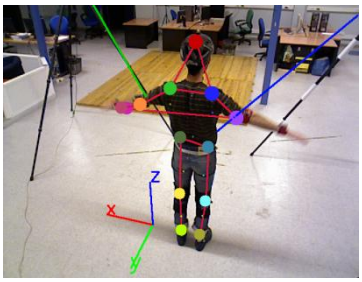

Figure 4-5 : Différenciation entre gauche et droite : ambiguïté.

La figure 4-5 illustre une limitation bien connue, notamment chez Shotton et al. Il s'agit de la non différenciation gauche-droite par les vecteurs de descripteurs. Si un descripteur met en jeu un voisinage restreint qui couvre à peine le volume d'un bras alors il ne pourra pas faire la différence entre le bras gauche et le bras droit dans certaines situations. Et dans le cas contraire, si le descripteur a un voisinage plus important (dimension de tout le corps), il permet de mieux différencier le bras gauche et droit dans certains cas, mais perd la propriété de localité. Cette propriété de localité est préservée si, indépendamment de la posture, la description du bras reste identique pour faciliter son apprentissage. Il faut donc retenir le bon compromis pour la taille du voisinage du descripteur (étudié en section 3.5.6).

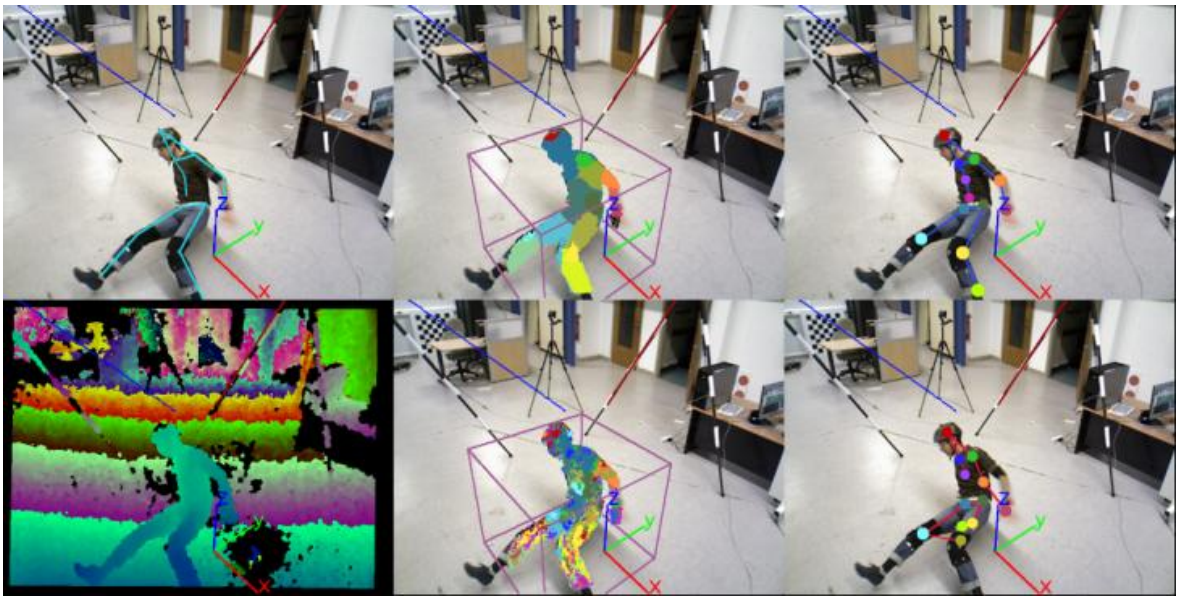

Figure 4-6 : Posture complexe, avec erreur de reconstruction.

La figure 4-6 représente un exemple de mauvaise reconstruction due à une posture intentionnellement complexe. Notre base de données contient volontairement un grand nombre de postures complexes pour éprouver notre approche. Ici, l'épaule gauche est correctement prédite malgré le faible nombre de voxels ayant le label associé. C'est grâce aux voxels internes qui participent à la classification même s'ils ne sont pas visibles à la surface des différentes vues.

Des vidéos illustratives sont accessibles via le lien URL :

http://www.wassfila.com/BPR

4.1.4 Évaluations quantitatives sur la classification

À partir de la vérité-terrain de chaque voxel, et après l'apprentissage et la prédiction/classification (réalisé avec validation croisée en trois tiers), on évalue le taux de classification des voxels :

$$taux\ de\ classification\ des\ voxels = 66\ \%$$

Les trois groupes de l'apprentissage (issus de la cross validation) ont un résultat presque identique (65 %, 66 % et 66 %). Ce qui nous permet par la suite de caractériser les performances sur un seul de ces groupes et incluant 348 postures. Les postures, de complexité variable, induisent des taux variables de classification de voxels comme illustré sur la figure 4-7 : 22 % des postures ont un ratio de classification à 0,7, 20 % des postures ont un taux de 0.9, etc.

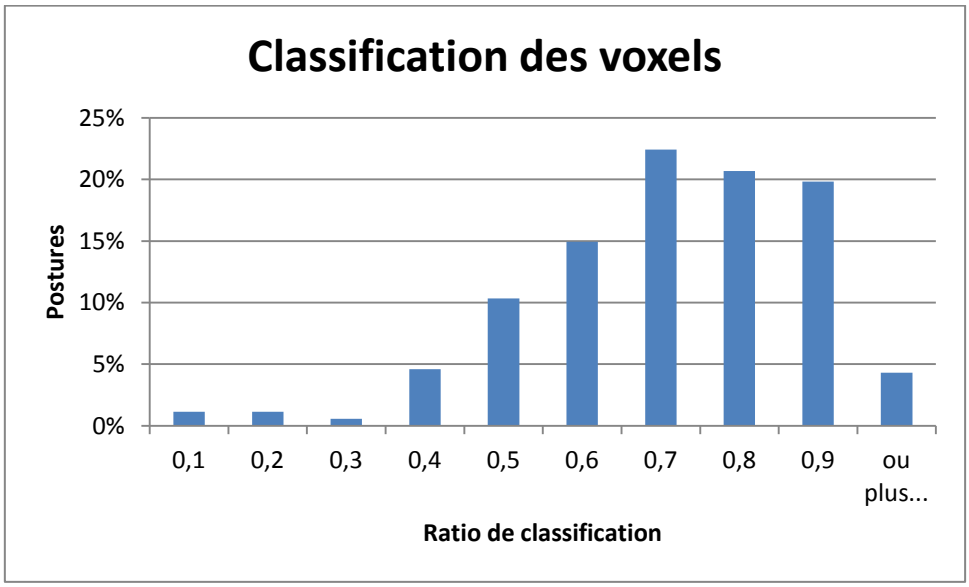

Figure 4-7 : Pourcentage des postures vs. ratio de classification des voxels.

Ces taux de classification sont évidemment à corréler avec la reconstruction des centres des parties corporelles. Avec l'algorithme du « Mean shift », seulement 1 % de voxels bien classifiés pourrait suffire pour inférer une position correcte du centre à prédire. Par ailleurs, 100 % de voxels vrais positifs d'une partie peut donner une reconstruction erronée s'il y a beaucoup de faux positifs pour cette même classe. Comme espéré le « mean shift » est assez robuste aux erreurs de classification.

Le tableau 4-4 quantifie les performances de classification pour la posture M3-236 (Cf. figure 4-4) via la matrice de confusion associée. Enfin, le tableau 4-5 indique pour cette posture M3-236 la matrice de confusion complète de toutes les classes des parties corporelles y compris les classes intermédiaires. On rappelle que les classes intermédiaires ont pour rôle de remplir le vide entre les classes des parties corporelles principales. Même si on note un large déséquilibre entre ces deux types de classes, le taux de classification des parties intermédiaires n'a pas d'impact sur l'estimation de la position des centres des parties corporelles principales.

Head	RShoulder	RElbow	RWrist	LShoulder	LElbow	LWrist	RHip	RKnee	RFoot	LHip	LKnee	LFoot	Sommes	<<-- Classified As
0	0	0	1	5	5	0	0	1	17	0	2	33	181	Uncknown
487	0	0	0	0	0	0	0	0	0	0	0	0	493	Head
0	1215	0	0	3	0	0	0	0	0	0	0	0	1248	RShoulder
0	0	791	0	0	21	0	0	0	0	0	0	0	1144	RElbow
0	0	0	466	0	0	3	0	0	0	0	0	0	548	RWrist
0	1	0	0	978	0	0	0	0	0	0	0	0	1037	LShoulder
0	0	3	1	0	646	0	0	0	0	0	0	0	693	LElbow
0	0	0	6	0	4	346	0	0	0	0	0	0	529	LWrist
0	0	0	0	0	0	0	1468	1	0	17	0	0	1531	RHip
0	0	0	0	0	0	0	0	1139	0	0	5	0	1230	RKnee
0	0	0	0	0	0	0	0	0	734	0	0	5	762	RFoot
0	0	1	0	0	0	0	12	0	0	1353	0	0	1410	LHip
0	0	0	0	0	0	0	0	5	0	0	1134	0	1263	LKnee
0	0	0	0	0	0	0	0	0	1	0	0	1146	1201	LFoot
1011	2109	1032	908	1551	1067	594	2356	1489	863	2107	1457	1294	Sommes	

Tableau 4-4 : Matrice de confusion des parties corporelles principales pour la posture M3-236.

Head	RShldr	RElbw	RWrst	LShldr	LElbw	LWrst	RHp	RKn	RFt	LHp	LKn	LFt	RS-RE	RS-LS	RE-RW	LS-LE	LE-LW	RH-LH	RH-RK	RK-RF	LH-LK	LK-LF	Sum	< Classified As
0	0	0	1	5	5	0	0	1	17	0	2	33	1	10	2	1	4	2	7	15	22	53	181	Uncknown
487	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	493	Head
0	1215	0	0	3	0	0	0	0	0	0	0	0	20	5	0	4	0	1	0	0	0	0	1248	RShoulder
0	0	791	0	0	21	0	0	0	0	0	0	0	90	0	225	11	6	0	0	0	0	0	1144	RElbow
0	0	0	466	0	0	3	0	0	0	0	0	0	0	0	79	0	0	0	0	0	0	0	548	RWrist
0	1	0	0	978	0	0	0	0	0	0	0	0	2	14	0	42	0	0	0	0	0	0	1037	LShoulder
0	0	3	1	0	646	0	0	0	0	0	0	0	1	0	14	10	18	0	0	0	0	0	693	LElbow
0	0	0	6	0	4	346	0	0	0	0	0	0	0	0	2	0	171	0	0	0	0	0	529	LWrist
0	0	0	0	0	0	0	1468	1	0	17	0	0	0	0	0	0	0	35	6	0	4	0	1531	RHip
0	0	0	0	0	0	0	0	1139	0	0	5	0	0	0	0	0	0	0	26	53	3	4	1230	RKnee
0	0	0	0	0	0	0	0	0	734	0	0	5	0	0	0	0	0	0	0	22	0	1	762	RFoot
0	0	1	0	0	0	0	12	0	0	1353	0	0	0	0	0	0	0	29	4	0	11	0	1410	LHip
0	0	0	0	0	0	0	0	5	0	0	1134	0	0	0	0	0	0	0	3	6	14	101	1263	LKnee
0	0	0	0	0	0	0	0	0	1	0	0	1146	0	0	0	0	0	0	0	14	0	40	1201	LFoot
0	291	122	3	19	11	0	0	1	0	12	0	0	4980	230	17	90	2	67	0	0	6	0	5851	RS-RE
522	591	0	0	322	15	3	3	0	0	3	0	0	663	19678	8	545	2	584	7	0	6	0	22952	RS-LS
0	0	52	421	2	24	16	0	0	0	0	0	0	8	0	1206	0	72	12	0	8	0	0	1821	RE-RW
2	11	40	0	215	217	1	1	0	0	1	0	0	154	155	11	6689	128	40	22	1	0	1	7689	LS-LE
0	0	0	5	0	94	225	0	0	0	0	0	0	1	0	51	2	793	0	0	0	0	0	1171	LE-LW
0	0	20	4	7	29	0	784	9	1	694	54	13	314	914	111	623	54	23888	892	16	714	39	29180	RH-LH
0	0	0	0	0	0	0	83	145	0	6	11	0	0	1	0	1	0	57	9815	204	551	71	10945	RH-RK
0	0	2	1	0	1	0	4	150	107	0	25	10	4	0	0	0	0	4	89	5317	61	115	5890	RK-RF
0	0	1	0	0	0	0	1	36	0	21	160	0	0	0	0	0	0	93	273	61	7957	160	8763	LH-LK
0	0	0	0	0	0	0	0	2	3	0	66	87	0	0	0	0	0	0	26	40	38	3020	3282	LK-LF
1011	2109	1032	908	1551	1067	594	2356	1489	863	2107	1457	1294	6238	21013	1726	8018	1250	24812	11170	5757	9387	3605	110814	

Tableau 4-5 : Matrice de confusion pour la posture M3-236.

On remarque, d'après cette matrice de confusion, que beaucoup de voxels mal classés appartiennent à une classe voisine, ce qui minimise considérablement l'impact sur l'erreur de

position. Néanmoins, au vu de la grande taille des parties intermédiaires, il n'est pas possible de conclure sur l'influence de cette erreur de classification.

Après ces observations sur les taux de classification, il faut maintenant considérer la métrique sur l'estimation des positions des centres des parties pour caractériser les performances de reconstruction.

4.1.5 Évaluations quantitatives sur les postures reconstruites

Nous privilégions logiquement la métrique proposée par Shotton et al. dans (Shotton et al. 2011a). Cette métrique quantifie l'erreur de distance entre la vérité-terrain et l'estimation de la position du centre des parties corporelles. On obtient une moyenne par posture, et une moyenne pour toute la base de données.

Nombre de centres évalués	4360
Nombre de centres trouvés	4029
Ratio des centres trouvés	0.92
Moyenne de l'erreur des centres trouvés	4.3 cm

Tableau 4-6 : Statistiques sur l'estimation des centres des parties corporelles.

Il est impossible d'incorporer dans la moyenne, la position des centres des parties corporelles qui n'ont pas du tout été trouvées. Il est donc judicieux de considérer une autre métrique qui est de comptabiliser plutôt le nombre de centres prédits ou mAP (pour « mean Average Precision) inférieur à un seuil de précision fixé a priori à l'instar de (Shotton et al. 2011a). Ainsi, le critère mAP est calculé pour plusieurs valeurs de seuils comme illustré par le tableau 4-7.

Seuil	<0.1	<0.2	<0.3	<0.4	<0.5
mAP	0.84	0.881	0.905	0.915	0.918

Tableau 4-7 : Pourcentage des centres prédits (mAP) avec une erreur inférieure au seuil.

On note ici que 84 % des centres ont une estimation dont l'erreur est inférieure à 10 cm.

Intéressons-nous à ces statistiques, partie par partie, par exemple sur la posture la M3-236 comme précédemment.

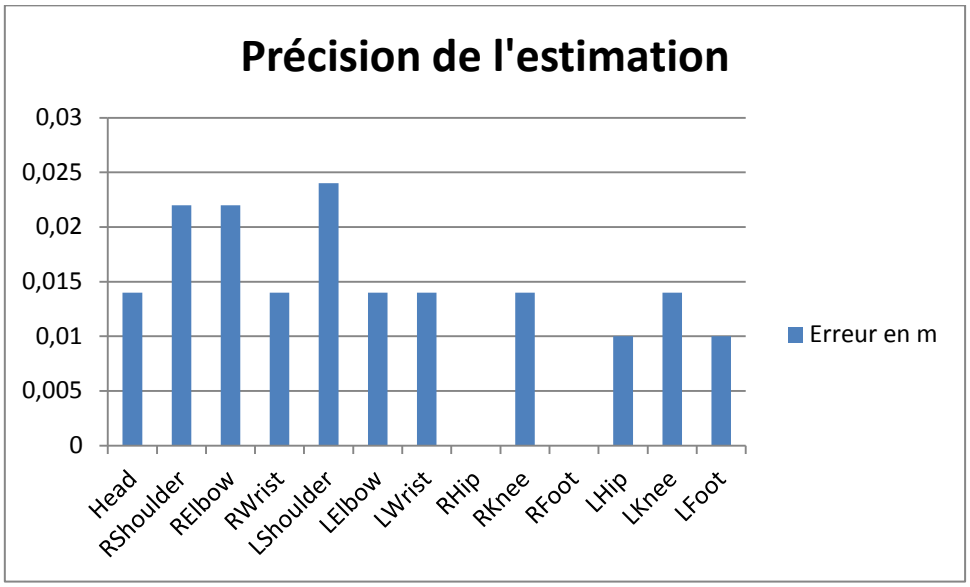

Figure 4-8 : Précision de l'estimation pour la posture M3-236.

La figure 4-8 représente l'erreur pour chaque centre de partie du corps. Ainsi, la tête donne lieu à une erreur de 1.3 cm sur sa localisation, etc.

On remarque que l'erreur de localisation est faible relativement à la taille des parties corporelles à localiser qui est de 10 cm.

4.2 Notre approche vs. OpenNI

On compare ici notre approche avec OpenNI en terme de précision de reconstruction. Il est difficile de mener la comparaison au-delà, car les applications commerciales comme OpenNI ne fournissent pas les codes sources ni les bases de données sur lesquelles l'apprentissage a été effectué. Certes, notre approche multi-capteurs doit logiquement supplanter OpenNI qui est mono-capteur mais il est opportun de quantifier les gains, etc. On focalise ici sur une séquence de test qui n'a pas été apprise lors de notre apprentissage. On rappelle que cette séquence utilisée est un ensemble de mini-séquences incluant des mouvements très diversifiés.

4.2.1 Paramètres d'apprentissage

On considère ici deux séquences ayant des mouvements similaires ; l'une a été utilisée pour l'apprentissage et l'autre pour le test comme indiqué sur le tableau 4-8.

Base de données

	Apprentissage	Test
Base de données	IRSS35-C2	IRSS35-C3
Nb Postures	1829	1814
Nb voxels |S|	112 M	Non appris
Mouvements	*Posture en T, haltérophilie, tennis, volley ball, ping-pong, natation (bras), pétanque, lancement de poids*	

Tableau 4-8 : Étude comparative : ases de données utilisées.

Paramètres d'apprentissage

Le tableau 4-9 énumère les paramètres libres et leurs valeurs associées pour cette étude.

Nombre de Candidats par nœud (d')	1500
Ratio de sous-échantillonnage (α)	0.3
Profondeur max (D)	500
Limite du gain d'information (t)	0.01
min norme vecteur descripteur (ϕ_{min})	0.02
Max norme vecteur descripteur (ϕ_{max})	0.7
Moyenne norme vecteur descripteur (ϕ_A)	0.36
Nombre d'arbres (N)	3
Min échantillons par nœud (m)	100

Tableau 4-9 : Étude comparative : valeurs des paramètres libres.

Le temps d'apprentissage pour ces 3 arbres a été de : ***7h 17mn***

4.2.2 Observations sur la reconstruction par OpenNI

Figure 4-9 : Évaluation qualitative d'OpenNI.

Rappelons que OpenNI reconstruit la posture pour chaque vue indépendamment i.e. chaque capteur a sa propre chaîne de labellisation sur l'image de profondeur puis de reconstruction. La démarche semble parfois cohérente dans la vue du capteur, mais les erreurs sont mises en évidence lorsque on reprojette la posture sur les autres points de vue.

Ainsi, la figure 4-9 montre une ligne constituée des 3 vues. Elle permet une visualisation des erreurs de reconstruction du squelette par OpenNI.

On représente en rouge (respectivement vert ou bleu) le squelette inféré depuis le capteur 1 (respectivement le capteur 2 ou 3).

L'image haut-gauche est la vue du capteur 1, dans laquelle est projeté le squelette reconstitué par le capteur 1 en plus du squelette de référence (vérité terrain) en cyan. Les deux autres vues de la ligne affichent le squelette de référence également, en plus du même squelette du capteur 1 projeté sur les vues correspondantes.

L'image au centre (2eme ligne, 2eme colonne) est la vue du capteur 2, dans laquelle est projeté le squelette reconstitué par le capteur 2 en plus du squelette de référence en cyan.

L'image en bas à droite est la vue du capteur 3, dans laquelle est projeté le squelette reconstitué par le capteur 3 en plus du squelette de référence en cyan.

Ces figures montrent que OpenNI n'a généralement de posture cohérente que dans la vue du capteur en question, d'où la limitation de l'approche mono vue.

4.2.3 Comparaison qualitative

Cette section discute les performances relatives sur quelques exemples clés.

La figure 4-10 montre un exemple-type de comparaison entre notre reconstruction de squelette et celle de OpenNI.

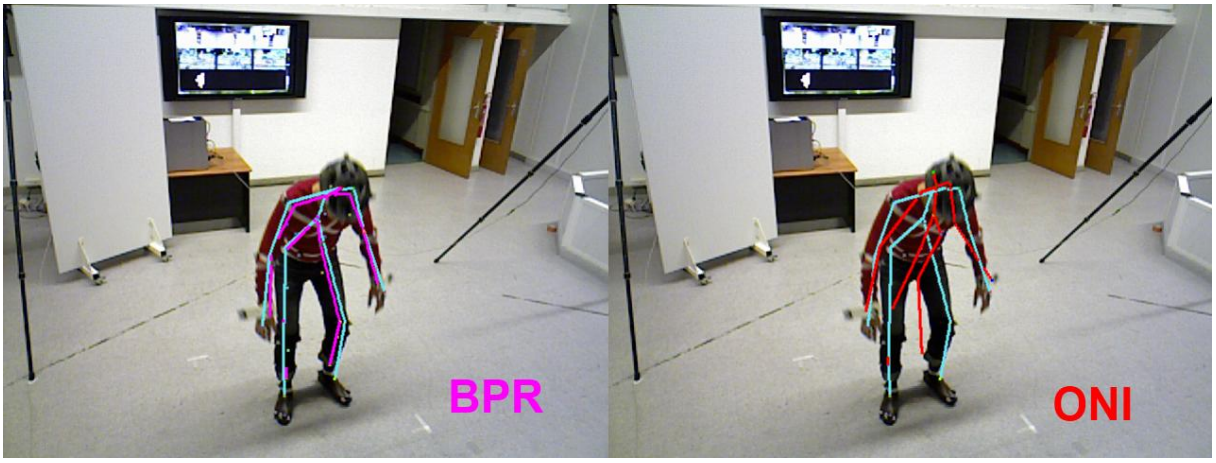

Figure 4-10 : IRSS35-C3 :BPR vs. OpenNI.

On remarque que OpenNI arrive à détecter plus facilement les extrémités des membres, mais présente plus de difficultés pour des parties intermédiaires e.g. les hanches.

Des exemples autres sont listés ci-après. On illustre le résultat de notre approche (notée **BPR)** en haut et en couleur rose, et openNI avec le capteur 0 (**ONI0**) en rouge.

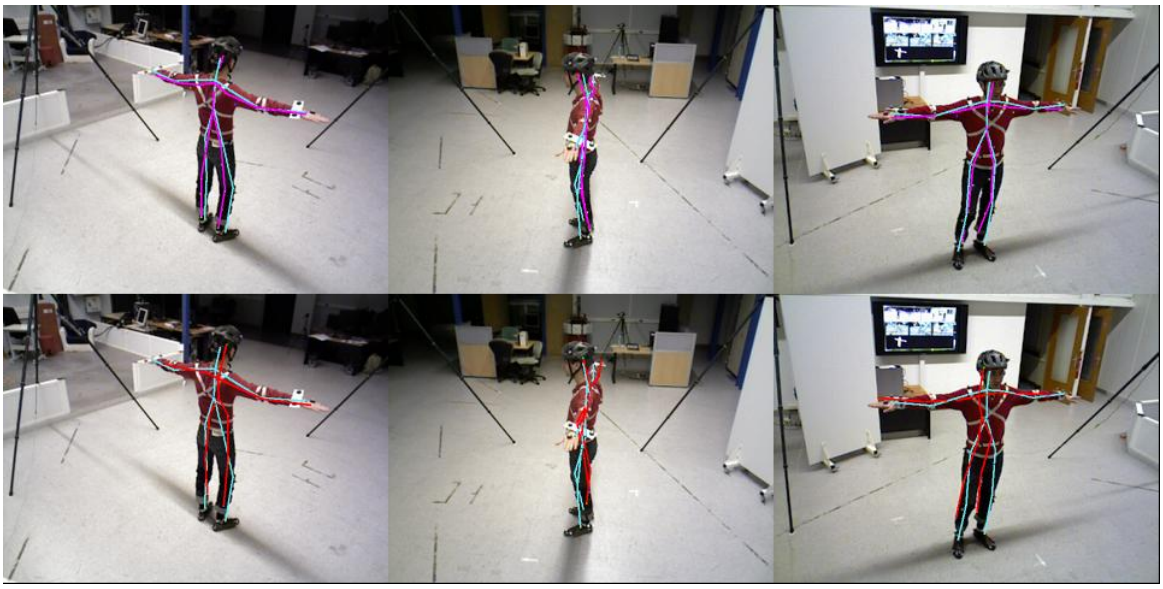

Figure 4-11 : Posture simple – Posture en T (IRSS35-C3-121).

La figure 4-11 présente une posture simple à reconstruire. Sur l'image bas-gauche, ONI0 semble visuellement correct, alors que projeté depuis les autres vues, on perçoit mieux l'erreur de localisation. C'est une erreur typique due à l'utilisation d'un mono capteur ; cette erreur n'est pas observée par notre approche multi-capteurs.

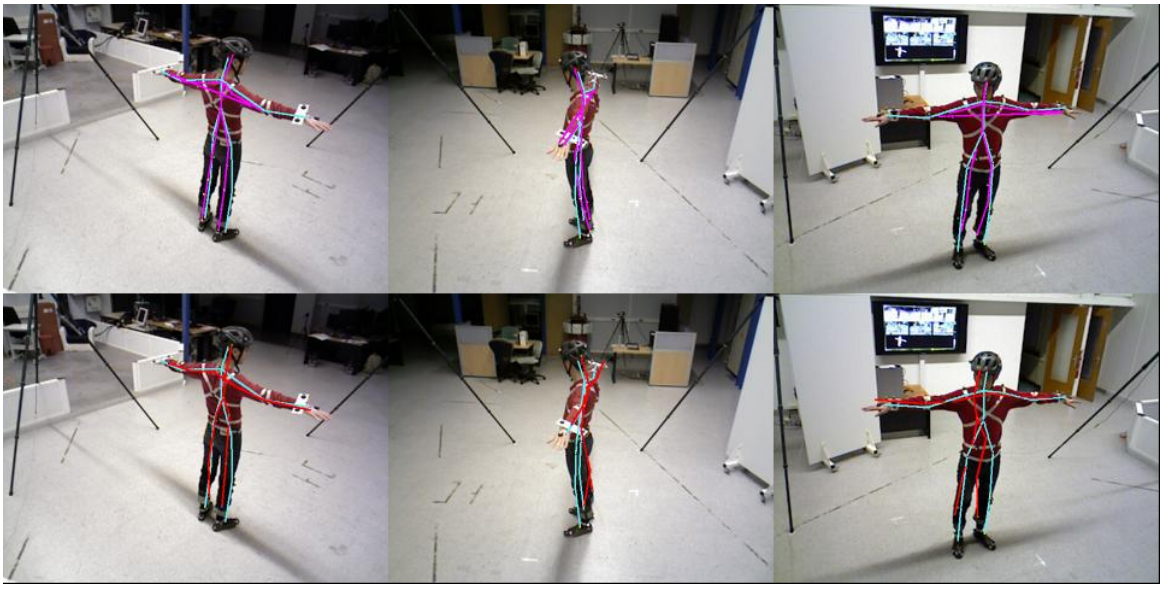

Figure 4-12 : Erreur de « flip » gauche-droite (IRSS35-C3-147).

La figure 4-12 présente le phénomène d'erreur de classification entre parties du corps symétriques (Cf. base de données NSC13 dans 4.1.3) et notée « flip ». Cette erreur est observée dans l'image haut-droite où la main droite du squelette s'est logée dans la même position que la main gauche. Cela est dû à la similitude des parties du corps apprises. Cet effet est prononcé quand le voisinage ne permet pas de les différencier. Quand le corps est en posture T, il a un axe de symétrie et rend les mains difficilement différentiables. ONI0 n'a pas cette erreur, ce qui nous permet de conclure qu'il contient un filtrage dans le processus de traitement.

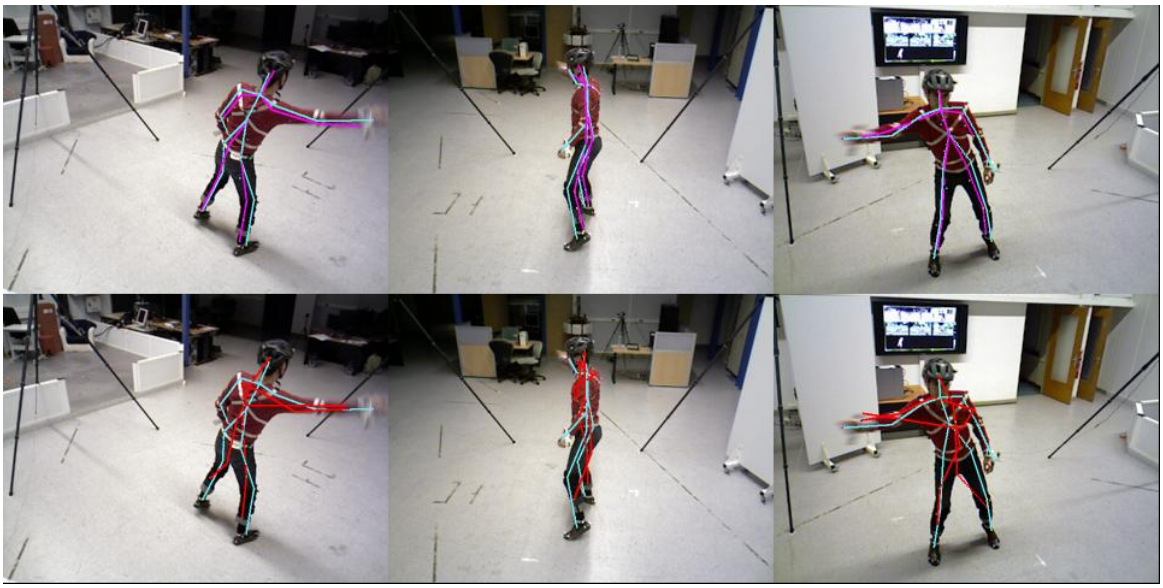

Figure 4-13 : Fusion de la main (IRSS35-C3-473).

La figure 4-13 illustre une posture où la main gauche est collée au corps. Aucune des deux approches n'arrive à placer correctement la main gauche, mais celles-ci se comportent différemment. ONI0, en l'absence de main gauche potentielle, a opté pour une posture plausible où les deux mains sont confondues avec la main droite. Notre approche arrive à placer correctement le coude gauche, et décide qu'il n'y a pas assez de voxels labellisés en main gauche pour pouvoir la prédire correctement ; celle-ci est donc absente de la reconstruction.

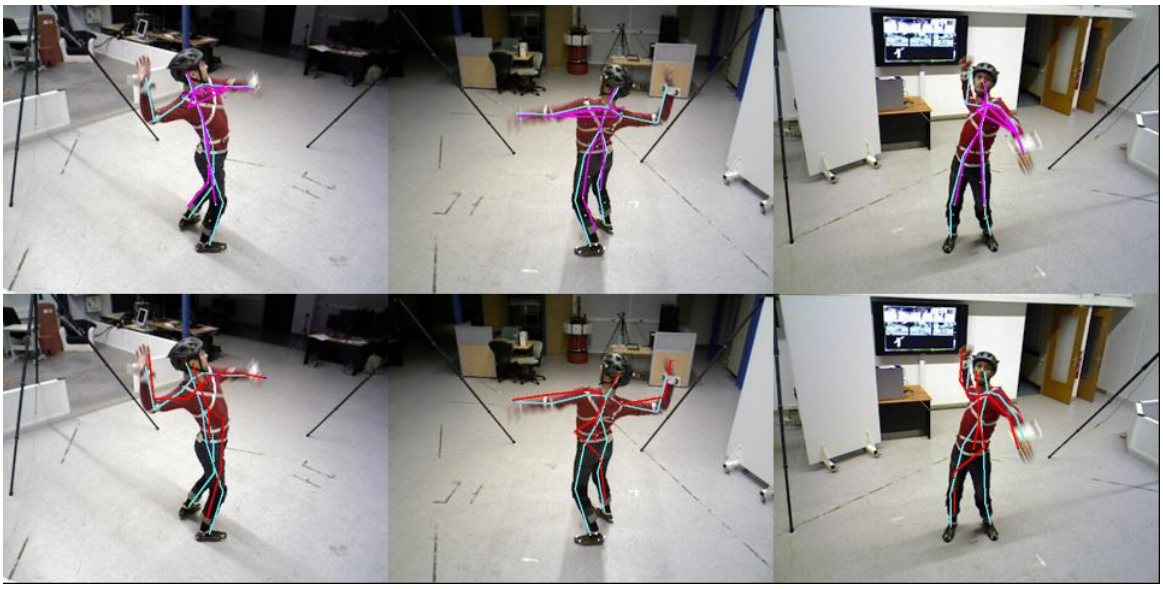

Figure 4-14 : Fusion des mains BPR (IRSS35-C3-917).

La figure 4-14 illustre une posture où notre approche a confondu les deux mains en une seule alors que ONI0 arrive à détecter le haut du corps correctement. Pour le bas du corps, notre approche ne détecte pas les pieds alors que ONI0 confond la jambe gauche avec la droite. On remarque d'après ces deux dernières figures que chaque approche a ses avantages et ses inconvénients pour la détection.

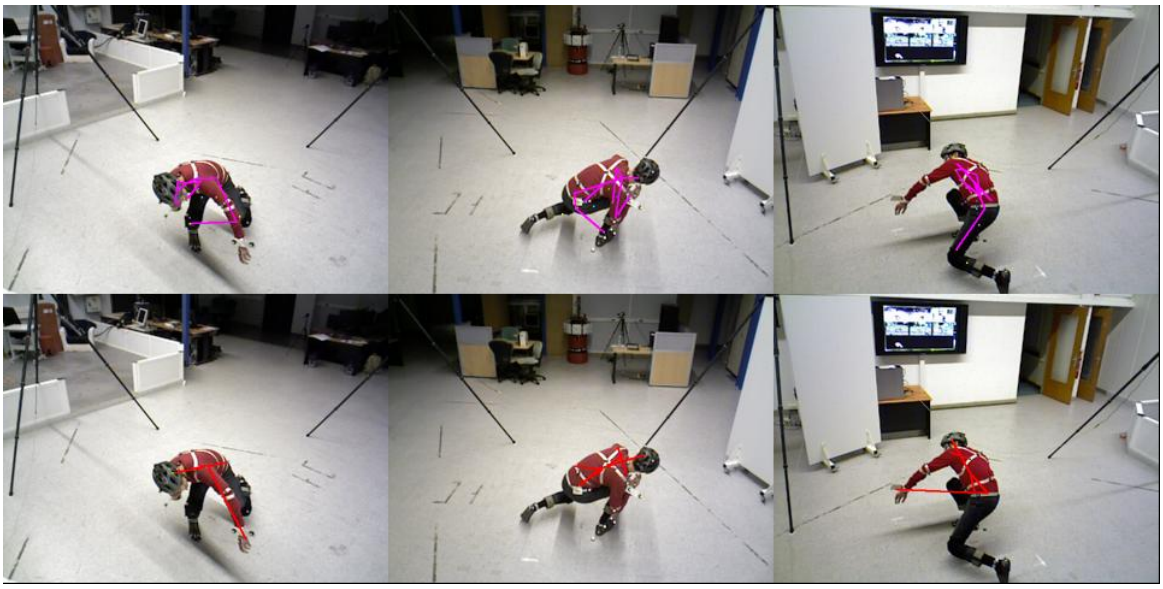

Figure 4-15 : Posture complexe (IRSS35-C3-1475).

La figure 4-15 illustre une posture complexe. Aucune des approches n'arrive à un résultat satisfaisant. Ceci illustre les limites de notre approche en présence de fortes auto-occultations. La main droite à titre d'exemple n'est visible que par une seule vue et elle est collée au corps et à la tête.

Au-delà de ces observations qualitatives, la section suivante montre des évaluations quantitatives sur le critère mAP.

4.2.4 Comparaison quantitative

Le critère mAP est estimé pour des seuillages différents, allant de 0,01 m à 0,5 m. À 0,5 m le seuil n'est plus significatif pour localiser une partie du corps. Le seuil le plus pertinent est autour de 0,1 m.

La figure 4-16 affiche en rouge, vert et bleu les courbes d'évaluation des squelettes issus des trois capteurs traités par OpenNI. Ces courbes sont comparées à notre approche (toujours notée BPR).

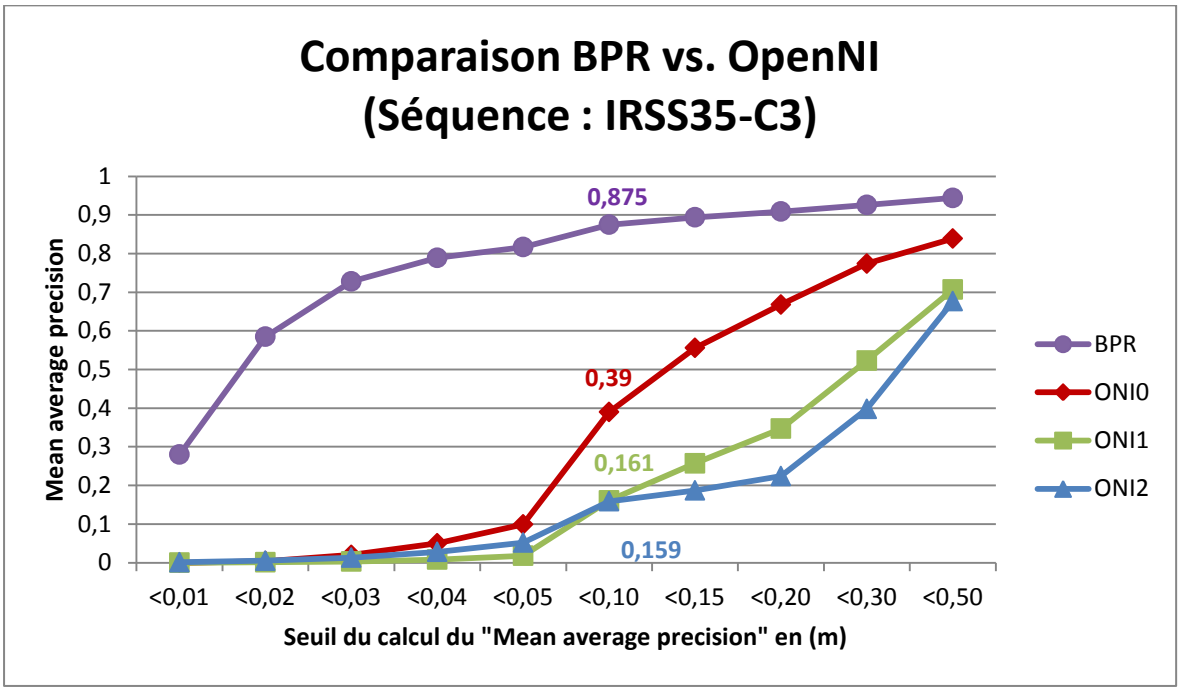

Figure 4-16 : Comparaison quantitative : notre approche vs. OpenNI (séquence 1).

Notre approche supplante OpenNI, et ceci dans toute la gamme de précision du critère mAP.

Cette comparaison quantitative est appliquée à une seconde paire de séquences présentées dans le tableau 4-10.

	Apprentissage	Test
Base de données	IRSS35-C4	IRSS35-C5
Nb Postures	1 068 dont 722 utilisés	1 327
Nb voxels appris |S|	44 M	Non appris
Mouvements	Posture en T, Tour sur soi, Marche, Course, Assis debout, Assis par terre, Accroupi	

Tableau 4-10 : Seconde base d'étude supplémentaire.

Les paramètres d'apprentissage sont les mêmes que ceux qui sont présentés dans le tableau 4-9.

Dans la figure 4-17, on remarque l'absence du signal du squelette d'OpenNI au niveau du capteur 0 (ONI0). Cela a été dû à des ratées occasionnelles de génération du squelette qui ont été observées lors des acquisitions.

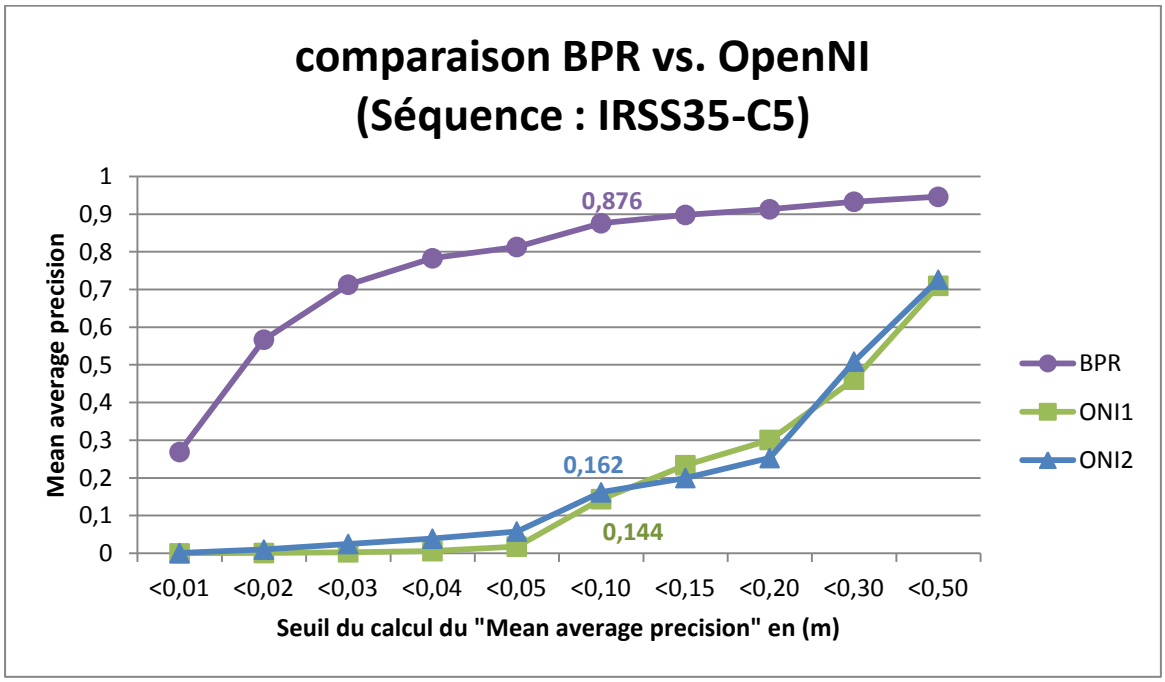

Figure 4-17 : Comparaison quantitative : BPR vs. OpenNI (séquence 2).

4.3 Conclusion

Ce chapitre compare et discute sur divers critères (classification, précision de reconstruction) les performances entre notre approche avec celle de Shotton et al. Les gains observés pour notre approche ont permis la réalisation d'une publication internationale (Filali et al. 2013).

Notre approche tire clairement parti de l'aspect multi-capteurs qui minimise les auto-occultations et permet donc de mieux gérer les postures complexes.

De plus, l'implémentation de notre approche par voxellisation lui confère un avantage relatif au temps de prédiction. L'utilisation de l'image de profondeur par (Shotton et al. 2011a) souffre d'un effet de perspective qui nécessite une correction appliquée au niveau de chaque pixel à classifier. En effet, dans une image de profondeur, il faut une transformation qui permet de passer de la mesure en pixels à la mesure en distance, cette transformation dépend de la profondeur du pixel en question. Notre approche en voxel dans l'espace cartésien n'a pas besoin de transformation, et un seul accès mémoire suffit pour évaluer un descripteur.

Au-delà des atouts algorithmiques, le chapitre suivant s'intéresse à l'intégration matérielle de notre approche i.e. il pose les jalons d'une architecture matérielle dévolue à notre approche.

Chapitre 5 : Intégration matérielle sur FPGA

Ce chapitre est plus particulièrement dédié à la présentation de quelques exemples d'intégration matérielle pour la détection de posture, lesquels prennent appui sur:

- les contraintes "temps réel" inhérentes au domaine de la robotique mobile, c'est à dire une cadence de prise de vues de 30 images par seconde.

- les avancées technologiques constatées au cours de ces dernières années (capteurs de profondeurs Kinect, GPU, ..) et qui ont pu remettre en question certains de nos choix.

- les différents travaux menés dans le groupe de recherche sur des activités connexes.

Enfin pour des questions d'embarquabilité et de réalisation de réseaux de caméras, une solution à base de FPGA sera en partie présentée. Dans cette dernière, seuls quelques aspects seront traités et évalués.

Ainsi, on trouvera dans les lignes qui suivent :

- un état de l'art des systèmes embarqués à architecture matérielle/logicielle qui nous guidera dans nos choix.

- une approche méthodologique basée sur la modélisation de haut niveau SysML facilitant la conception de tels systèmes largement complexes.

- un comparatif sur l'intégration de quelques fonctions sur différentes architectures.

5.1 État de l'art sur la méthodologie et l'architecture

La détection de posture est une application spécifique et gourmande en ressources. Par contre, selon le contexte de l'application, on peut avoir différentes contraintes issues de différents besoins. Les solutions qui répondent à ces différents besoins sont diverses et chaque environnement a son propre processus de conception. C'est pour cela que l'on commence par un état de l'art sur la méthodologie des systèmes embarqués (section 5.1.1), puis on répartit les architectures existantes selon notre vision (section 5.1.2). L'étape suivante est une analyse (section 5.1.3) qui nous permet d'évaluer la complexité de l'architecture nécessaire en fonction de la complexité de l'algorithme et des performances souhaitées. Ensuite, on présente et on commente un exemple d'une architecture existante (section 5.1.4). La suite fait appel à un processus de conception de type rencontre au point milieu « ***meet in the middle*** ». Il s'agit d'abord de modéliser le système à concevoir de façon globale. Pour cela, notre choix s'est porté sur différents diagrammes SysML (section 5.2). Ensuite, on part du niveau bas en détaillant la plate-forme qui nous a paru la plus appropriée pour une application « temps réel » et « embarquée » (section 5.3). C'est un environnement dédié à la conception de systèmes à base de circuits FPGA. On y contribue avec des composants permettant de faciliter le lien avec le haut

niveau, en levant le niveau d'abstraction (gestion des flux et non pas des instructions). La rencontre au point milieu se concrétise avec une convention de modélisation standard qui est le placement des fonctions. Il s'agit de répartir les tâches sur les différentes ressources matérielles disponibles (section 5.4). Cela est le cas pour les différentes solutions retenues. Dans la section 5.4.4, on compare la fonctionnalité de soustraction de fond implémentée sur les différentes architectures (PC, GPU, FPGA) afin de justifier les différents contextes d'utilisation de chacune des solutions proposées. Enfin on détaille l'implémentation de cette fonctionnalité sur FPGA qui est l'environnement qui nous intéresse.

5.1.1 État de l'art sur la méthodologie pour les Systèmes embarqués

Différentes approches peuvent définir un système embarqué. Le point de vue système : Le conseil international en ingénierie système (INCOSE[11]) définit la discipline comme une approche et des moyens interdisciplinaires permettant la réalisation de systèmes réussis.

Dans « Embedded Systems Design » (Heath 2002) un système embarqué est défini comme étant spécifique à une fonction donnée contrairement au PC dont la fonction est définie par un module logiciel. Cette définition datée de 2002 a précédé l'évolution du système embarqué type qu'est le « smart phone » dont le logiciel détient une partie prépondérante croissante.

Parmi les domaines d'application des systèmes embarqués, on peut citer :

- Transport et défense : Avionique, automobile, ferroviaire (motorisation, contrôle de commande, freinage, vision, sécurité)
- L'électroménager (four, réfrigérateurs,…)
- Téléphonie mobile (smart phones)
- Le multimédia (téléviseurs, caméras, équipements audio, consoles de jeux)
- Biomédical (ECG, analyseurs ADN,…)
- Équipement informatique (disques durs, lecteurs DVD) et tablettes.
- Outillages et équipements divers.

Avec la croissance de la complexité des systèmes embarqués, beaucoup se sont penchés sur la méthodologie adaptée à ces nouvelles contraintes. L'UML « Unified Modeling Language » (Langage de modélisation unifié) est couramment utilisé dans les projets logiciels. Avec sa possibilité de spécialisation en profils dédiés, on constate l'introduction de profils spécifiques aux systèmes temps réel puis aux systèmes embarqués.

(Martin et al. 2001) a présenté un profil appelé « Embedded UML » dont plusieurs traits sont hérités de l'UML temps réel et les concepts sont inspirés du co-design Matériel/Logiciel. Il a aussi appliqué une cartographie « mapping » qui transpose une conception UML pour générer une implémentation matérielle et logicielle exécutable sur une plate-forme spécifique. L'approche est très enthousiaste **prétendant que** contrairement aux générateurs UML qui créent un code non optimisé, celui-là a vocation de l'être en utilisant les bus et mémoires spécifiques, ainsi que les vraies interruptions et DMA (Direct Memory Access). **Par contre**, il indique que ceci ne sera pas automatisé au début, mais que l'utilisateur doit intervenir **manuellement** pour configurer le « mapping ».

[11] http://www.incose.org

(Martin 2002) apporte une vision plus réaliste et pertinente, il analyse les différents profils actuels de l'UML et conclut que de futurs travaux sur la méthodologie sont encore nécessaires avant de pouvoir appliquer l'UML au concept qu'il appelle « UML Platform » qui considère les spécificités des systèmes temps réel et systèmes embarqués.

L'adaptation a continué avec l'UML 2.0 avec lequel (Kukkala et al. 2005) a créé le profil TUT (avec TUT étant les initiales de « Tampere University of Technology »). Le profil TUT est un exemple parmi tant d'autres qui essaient d'introduire des règles de conceptions spécifiques à des applications ou des plates-formes. Bénéficiant de l'expertise chez Nokia[12], cela illustre la divergence d'un standard UML, au cas par cas. Chacun disposant de ses propres outils et méthodes, par-dessus lesquels il rajoute un profil UML qui lui reste spécifique.

Suite à ces différentes initiatives de variations à travers des profils, l'OMG[13] (Object Management Group) à la base de la spécification d'UML a collaboré avec d'autres groupes pour spécifier une extension d'UML appelée ***SysML*** adopté en 2005 avec la dernière version SysML 1.3 qui date de 2012.

Dans sa présentation[14] Pascal Roques affiche SysML comme un remplacement de la méthodologie SADT « Structured Analysis and Design Technique ».

Dans la conception des systèmes, on passe naturellement par plusieurs phases de modélisations avec des niveaux d'abstraction différents. De la simple idée exprimée en une phrase, au schéma explicatif avec des boîtes, restreint à une seule vue, et ainsi de suite jusqu'à l'implémentation finale qui est le système lui-même et non pas un modèle simplifié du système. On peut admettre dans ce cas que la modélisation fait partie de la conception du système. On remarque néanmoins, qu'à partir d'un certain niveau d'abstraction, on essaie de formuler une représentation unique du système. Cela évite d'avoir des divergences entre les différents niveaux d'abstraction. Le souhait est d'avoir idéalement des outils de modélisation, qui permettent de faire le lien entre ces différents niveaux d'abstraction et la conception en bas niveau, avec le bonus d'avoir le maximum d'automatisation et de génération de code et de structures. Ce souhait a fait l'objet de beaucoup d'efforts qui essaient de faire de l'outil de modélisation UML ou SysML, un outil de conception qui permet le passage transparent à l'implémentation. D'un autre côté, ceux qui ont de l'expérience avec les outils bas niveau comme le C, le C++ et les langages HDL (Hardware Description Langage) trouvent le passage direct du modèle à la génération du code contraignant à l'usage et mal adapté aux besoins d'optimisation.

D'autres alternatives émanant du cadre industriel comme à titre d'exemple chez STMicroelectronics[15], ont essayé de combiner différentes approches pour combler le vide entre les étapes de conception. (Riccobene et al. 2005) combine UML 2.0 avec le SystemC. Leur profil UML représente les caractéristiques structurelles et comportementales permettant une modélisation de haut niveau avec une traduction directe en SystemC. Le SystemC a pour cadre plus général le TLM (Transaction Level Modeling). (Frank Ghenassia 2005) explique dans ce livre l'intérêt et la place du SystemC. Après la révolution entre les années 1980 et 1990 qui élève l'abstraction du niveau de la schématique des portes logiques au niveau RTL (Register Transfer-Level), le SystemC est une initiative qui part du bas niveau pour élever l'abstraction RTL à un niveau supérieur. Après une décennie de recherche de niveau d'abstraction plus haut, le SystemC avec ces codes « open source » en C++ a émergé comme le moyen adéquat pour rapprocher les équipes de développement matériel et logiciel en un modèle de référence

[12] http://www.nokia.com
[13] http://www.omg.org
[14] http://www.irit.fr/SysML/lib/exe/fetch.php?media=journee_sysml_france:101206_desadtasysml.pdf
[15] http://www.st.com

unique. Cela introduit une réduction du temps de mise sur le marché et une amélioration de la qualité des systèmes embarqués.

Rappelons que le SystemC est défini par l'OSCI (Open SystemC Initiative). Même s'il est fondé sur des librairies en C++, il est considéré comme un langage en soi, avec des compilateurs et des simulateurs. Il a des similitudes avec les langages VHDL et Verilog, mais avec plus de complexité. Il est moins optimal pour la simulation RTL, mais il est en forte progression de version en version. Le dernier standard IEEE 1666-2011 inclut l'abstraction des ports, de la communication, des processus dynamiques et des notifications d'événements.

Le SystemC a pour principale vocation la modélisation des systèmes et la vérification fonctionnelle en comparant le SystemC avec le système final implémenté (DUT Device under Test en HDL Hardware Description Language). Un système modélisé en SystemC ne garantit en rien la possibilité de pouvoir l'implémenter sur telle ou telle plate-forme. L'implémentation est donc une autre problématique. Le SystemC joue aussi le rôle d'outil de synthèse de haut niveau, mais ceci reste restreint au même cadre de synthèse du code C en HDL qui est encore en phase d'étude expérimentale et en maturation. Même si des outils commerciaux existent pour de telles synthèses, ils sont hors de prix pour les petits et moyens budgets et leur usage est loin de couvrir toutes les subtilités de l'optimisation des langages HDL. Rappelons que cette complexité vient du contraste entre le C++ et les HDL. Tout programme C++ compilable peut s'exécuter, ce qui n'est pas du tout le cas des HDL pour lesquels il faut avoir une connaissance préalable des structures synthétisables.

En ce qui concerne non pas la modélisation mais la synthèse en haut niveau, les constructeurs de composants FPGA Altera et Xilinx disposent de leurs propres approches. Elle part principalement des modules matériels reconfigurables et permet à des outils de synthèse en haut niveau simplement de les interconnecter. Tous les cas particuliers sont pris en considération pour optimiser l'usage des ressources correspondant à la fonction souhaitée. Simulink sur Matlab nous paraît l'environnement qui fait le meilleur compromis entre facilité d'utilisation, conceptualisation du projet en imbrication de boîtes et optimisation de l'empreinte synthétisée automatiquement. Néanmoins, pour plus de flexibilité, on peut tout à fait se limiter aux outils de haut niveau tels que *SOPC Builder* pour Altera et *System Generator* pour Xilinx. Une dernière alternative en cours de maturation est l'utilisation de noyaux de calculs avec OpenCL. Une technique à la base adaptée pour les GPU multicores, le langage OpenCL lui a donné plus de flexibilité pour pouvoir l'implémenter tout aussi bien sur des architectures GPU que CPU multicores. Le constructeur Altera a étendu l'usage de ce langage pour l'exécuter sur une architecture FPGA.

5.1.2 Répartition des architectures des calculateurs

On s'intéresse maintenant plus au matériel et on va segmenter l'espace des calculateurs pour s'y référer plus facilement. Il existe dans la littérature beaucoup de classifications (Abd-El-Barr & El-Rewini 2004; Hennessy & Patterson 2012). On peut répartir les calculateurs selon :

- L'**architecture** : Harvard, Harvard Modifiée, von Neumann, ou autres en flot de données à usage plus spécifique et moins répandu[16].
- Le **jeu d'instructions** : principalement CISC, RISC avec d'autres variantes possibles.
- La **taille des mots** : de 1 à 256 bits, avec les plus répandus 8,16 bits pour les microcontrôleurs et 32 et 64 bits pour les ordinateurs communément utilisés.

[16] http://cnc.cs.manchester.ac.uk/projects/dataflow.html

- Le **parallélisme** : au niveau des instructions et/ou au niveau des données ensuite aussi au niveau des « threads » avec multiples unités de calcul[17].
- La **mémoire** : en fonction de sa bande passante, ses différents niveaux de caches hiérarchiques et de partage entre unités de calcul.
- L'**interconnexion** : dans le cas de réseaux de calculateurs pour des fins applicatives comme pour l'automobile, ou pour plus de puissance de calcul pour les centres de calcul.

Selon ces critères, il n'est pas possible de regrouper des ensembles de calculateurs de façon objective, car il existe plus d'exceptions que de règles.

On va alors procéder à une répartition qui dépend plutôt de l'usage. On l'illustre dans la figure 5-1.

On associe à PC, une unité de travail personnelle. Elle est constituée par un processeur de la famille x86. Elle permet l'échange de programmes compatibles entre tous ses utilisateurs. Un PC a longtemps été associé à des besoins énergétiques importants qui l'ont laissé en dehors des systèmes embarqués, même si actuellement on remarque une convergence entre les micromachines personnelles à base d'architectures ARM et la réduction de taille et de consommation des architectures x86 pour des applications embarquées.

Les GPU sont dans une catégorie à part vu que ce sont des systèmes non autonomes, qui ont besoin d'un PC pour la préparation et le lancement des tâches. L'utilisation de GPU commence à devenir une alternative pour les applications les plus coûteuses en temps de calcul et concurrence l'usage des grappes de serveurs. Par contre, les outils logiciels de GPU représentent encore un frein pour les utilisateurs vu qu'il faut une formation spécifique. Les GPU qui font partie de composants embarqués ne sont pas considérés dans cette classe-là.

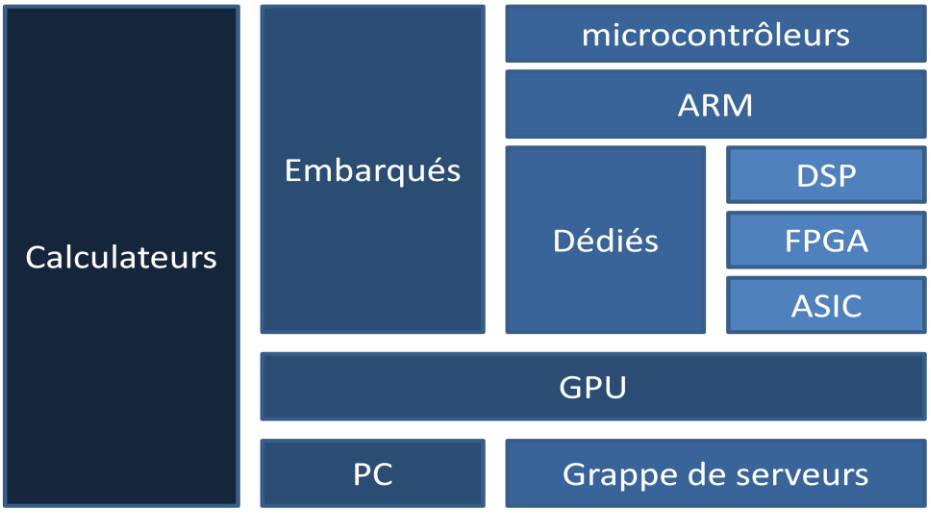

Figure 5-1 : Répartition des architectures des calculateurs.

Les FPGA et ASIC représentent un axe principal des systèmes embarqués, vu leurs caractéristiques d'adaptabilité et de possibilité de dimensionner les ressources. Par contre, les calculateurs, simulateurs et émulateurs à base de FPGA qui atteignent le maximum de performances n'ont pas d'objectif d'embarquabilité, et on les considère ici, juste par homogénéité.

[17] http://groups.engin.umd.umich.edu/vi/w3_workshops/VI_winter04_ganesan.pdf

La conclusion de cette partie est qu'une solution pour une application finale est souvent la combinaison de plusieurs calculateurs, c'est pour cela qu'il est important d'étudier l'application dans sa globalité avant de décider de la répartition et du choix de l'architecture matérielle. Il est aussi commun, comme dans notre cas, de développer des fonctions d'évaluation ou de simulation pour un prototypage sur PC avant de choisir l'architecture qui apporte les gains recherchés en performance, en consommation ou en coût.

5.1.3 Analyse des différentes architectures

L'approche adoptée pour développer ce projet, consiste à commencer par définir l'algorithme en fonction de l'objectif en s'inspirant de l'état de l'art, ensuite, à réaliser une première version implantée dans un environnement PC. Cette première version doit assurer une exécution en temps réel tout en se permettant d'utiliser autant de ressources processeurs que nécessaire (PC dual processor, 8 cœurs). C'est une application développée en C++, avec des optimisations dans les structures de données pour en accélérer l'exécution :

- Classification d'un voxel avec parcours d'un arbre de décision binaire.
- Calcul du test d'un nœud de l'arbre qui est défini par un vecteur auquel correspond un offset dans la structure du nuage des voxels. Cet offset permet de calculer la caractéristique du test en un seul accès mémoire.

Quand il s'agit de mesurer la complexité d'une fonction implémentée, l'application tourne en un « thread » unique (monoprocesseur) et le temps d'exécution est noté comme référence. Cette mesure est valable pour une configuration de PC donnée : dans notre cas, il s'agit d'un système 64 bit avec double processeur Intel Xeon E5620 à 2.4GHz, 8Go de RAM (8x1Go DDR3-1333) réparties sur trois canaux de contrôleurs mémoire.

Critères de choix

Une fois que l'on adopte une architecture différente, un PC différent, un processeur embarqué, un GPU, un FPGA, ou une combinaison de processeurs hybrides, on remarque que les critères les plus importants à considérer sont :

- Le **nombre d'opérations** de l'algorithme et le nombre d'instructions compilées sur le système cible.
- L'**interdépendance des fonctions** et l'interdépendance des opérations de la même fonction.
- L'**interdépendance des données** entre les fonctions et pendant l'exécution de la même fonction.
- La **puissance de calcul** nécessaire pour assurer le temps réel. Le temps réel étant d'atteindre les objectifs temporels de l'application.
- L'**équilibrage** entre la bande passante mémoire et la puissance de calcul pour une bonne utilisation des ressources. On remarque que selon l'algorithme, il est possible de recalculer les valeurs des variables intermédiaires ou de les stocker en mémoire et les récupérer. Cela permet de balancer entre la puissance de calcul et la bande passante. Une fois que l'architecture est spécifiée, cela permet d'y ajuster l'algorithme.
- La **parallélisation** : si l'application est parallélisée, il est souhaitable de bien définir le niveau de parallélisation. S'il se situe au niveau fonctionnel où chaque unité de calcul assure une fonction, ou s'il se situe au niveau des données, où chaque pixel est calculé par un processeur différent. Dans ces différents cas, on remarque que le temps de

changement « overhead » est différent en fonction de l'architecture matérielle. Cet « overhead » rend l'architecture appropriée soit à une parallélisation de fonctions en haut niveau soit à une parallélisation des données au bas niveau.

Ensuite, on a des critères un peu plus généraux qui dépendent du contexte de l'application :

- L'**embarquabilité** : si le système final peut tourner sur un serveur, un PC d'un utilisateur, intégré dans un système complexe comme une voiture ou un système de vidéosurveillance, ou un système portable sur batterie de dimension minimale. Quand le système doit faire partie d'un autre système plus complexe, on peut avoir envie soit de réduire la complexité du système central en assurant le maximum de fonctionnalités par les périphériques, soit de minimiser les fonctionnalités des périphériques pour profiter au maximum de la puissance de calcul du système central.
- La **consommation** : qui est une contrainte héritée de l'embarquabilité, principalement si l'appareil est sur une batterie et de taille minimale. Et qui d'un autre côté doit satisfaire la contrainte temporelle de l'application qui exige d'opérer les instructions en un temps imparti.
- Le **coût** : selon le coût, plusieurs technologies peuvent être considérées, si la puissance de calcul le requiert. Il est aussi important de considérer le coût du développement et de la production.
- L'**échelle du marché** : cela dépend s'il s'agit de développer une étude de faisabilité, une simulation, une application qui assure les contraintes temporelles, un prototype pour une application à usage exclusif, un prototype en phase avancée d'une grande série ou un produit destiné à un marché restreint ou à un marché grand public. Cette échelle du marché permet en fonction de sa taille, d'investir plus sur le développement pour minimiser les coûts à l'unité, ou inversement de minimiser les temps de développement avec des unités de calcul disproportionnées et des algorithmes et fonctions non optimisées. Ces unités de calcul ont l'avantage d'être assez génériques et réutilisables dans les centres de recherche. Tandis que pour un produit commercial, une petite faille pourrait nécessiter un retour de produits, ce qui engendre des coûts et une atteinte à l'image de la marque.

Tableau récapitulatif

Le tableau qui suit permet de donner des indications sur les ressources disponibles sur différentes plates-formes de développement. La performance de calcul est purement indicative et ne sert que pour une orientation ou pour définir l'ordre de grandeur des performances.

Processeur	Fréquence	Calcul	RAM	Unités de calcul
Microcontrôleur 8bit (PIC18F[18])	64 MHz	5 à 16 MIPS	16KB (data)	1
Microcontrôleurs 16bit (dsPIC33F[19])	340 MHz	70 MIPS	48 KB (data)	1
Processeurs embarqués 32 bits (ARMv7 Cortex-A9[20])	1GHz	2500 DMIPS/core	64 KB L1 8 MB L2 Up to 4GB sys	2
ARM + DSP (TMS320DM8148[21])	ARM A8 1GHz DSP 700 MHz	ARM: 2000 MIPS + DSP: 6000 MMACS	64K L1 ARM 64K L1 DSP 512 KB L2 ARM 256 KB L2 DSP DDR3-800 sys	2
FPGA Altera StratixV	500 MHz (DSP blocks)[22]	1840 GMACS[23]	50 Mb internal 6xDDR3-1066	3926 DSP blocks (18x18)[24]
FPGA Xilinx Virtex7	741 MHz (DSP Slice)	5335 GMACS	67 Mb internal 8xDDR3-1866	3600 DSP Slices (25x18)[25]
ARM + FPGA (Zynq-7000[26])	ARM A9 dual core 1GHz + logic	2500 DMIPS/core + 1334 GMACS	32KB L1 512KB L2 + 19Mb internal	2 cores CPU + 900 (25x18 MACCs)[27]
GPU (nVidia Tesla K20X[28])	732MHz	3950 GFLOPS	6 GB Bandwidth : 250 GB/s	1 GK110 with 2688 cores
PC (HPZ800[29])	3.06 GHz	144.88 GFLOPS[30]	12MB cache L3 Up to 192 GB (12x16GB) DDR3-1066	2x6 cores 24 threads

Tableau 5-1 : Ressources de différentes plates-formes.

Les processeurs de faible puissance sont référencés par MIPS (Million d'instructions par seconde), sur la même échelle, on retrouve les MMACS (Million de multiplications et Accumulation par seconde), ensuite des Giga MACS. Pour les processeurs plus performants

[18] http://www.microchip.com/pagehandler/en-us/family/8bit/architecture/home.html
[19] http://www.microchip.com/pagehandler/en-us/family/16bit/architecture/dspic33f.html
[20] http://arm.com/products/processors/cortex-a/cortex-a9.php?tab=Specifications
[21] http://www.ti.com/product/tms320dm8148
[22] http://www.altera.com/devices/fpga/stratix-fpgas/about/dsp/stx-dsp-block.html
[23] http://www.altera.com/b/stratix-v-fpga.html
[24] http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp
[25] http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
[26] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm
[27] http://www.xilinx.com/publications/prod_mktg/zynq7000/Zynq-7000-combined-product-table.pdf
[28] http://www.nvidia.com/content/tesla/pdf/Tesla-KSeries-Overview-LR.pdf
[29] http://h18004.www1.hp.com/products/quickspecs/13278_na/13278_na.pdf
[30] 3.06 GHz × 6 cores × 4 inst per cycle × 2 CPU = 144.88 GFLOPS

ayant des unités de calcul flottantes, l'unité est le GFLOPS (Giga opérations de virgule flottante par seconde).

Dans le cas le plus simple, la mémoire est unique et générale. Dans les autres cas, on trouve des mémoires caches de plusieurs niveaux vu que les mémoires DDR ont une grande bande passante mais un temps de latence très élevé comparé aux mémoires caches.

Concernant les unités de calcul, on s'y réfère pour indiquer le nombre d'opérations qui peuvent s'exécuter en parallèle indépendamment de la capacité de parallélisation de l'application. Dans les FPGA, ce sont des blocs DSP connectés au reste de la logique programmable. Chaque bloc peut constituer un processeur ou une autre unité de traitement. Pour les GPU, ce sont des unités de calcul qui sont regroupées, dont chaque groupe partage le même contrôleur d'instruction. La parallélisation de l'application a donc un impact sur l'efficacité de l'architecture.

5.1.4 Exemple de matériel dédié à la détection de posture.

La détection de postures est une application de haut niveau. On va donc considérer les solutions matérielles qui répondent à cette fonctionnalité spécifique.

Kinect

Le kinect[31] représente la première solution mise à disposition du marché et la plus répandue. On peut y référer selon notre analyse comme un regroupement de calculateurs spécifiques à l'application avec une composante embarquée et une composante fixe. La figure 5-2 représente la chaîne applicative du kinect présentée avec notre terminologie de la figure 5-1.

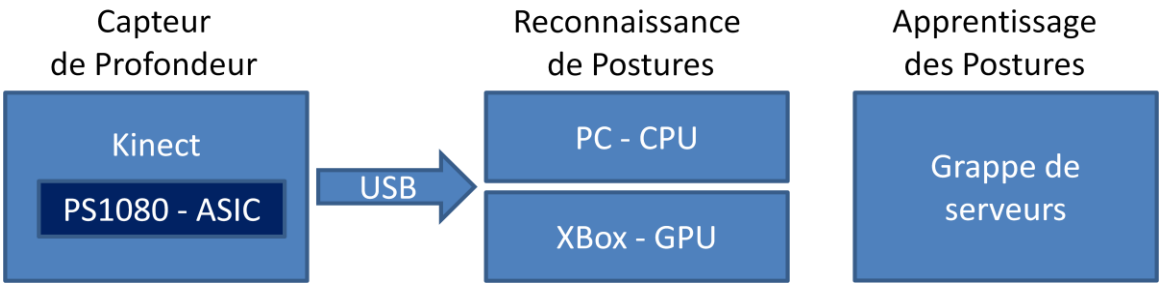

Figure 5-2 : Utilisation du kinect.

Le processus algorithmique implémenté dans le cadre applicatif du kinect est décrit dans (Shotton et al. 2011b) et présenté dans la section 2.1.4. Ce processus ne concerne que l'étape de la reconnaissance de postures. La parallélisation de l'application d'apprentissage des postures est décrite dans (Budiu et al. 2011). Il y a eu usage d'une grappe de serveurs comprenant 235 machines. L'apprentissage compte 3 arbres de profondeur 20, 31 parties du corps avec 300 000 images par arbre et 300 000 pixels pour chaque image qui sont sous-échantillonnés en 2 000 pixels. Le temps de calcul est évalué à 100 CPU-jours.

Concernant les processus qui sont exécutés en ligne, deux versions existent pour l'application de reconnaissance de postures : la version sortie en premier qui tourne sur le GPU de la console Xbox[32] qui a été suivie d'une version PC. La version PC, ne bénéficiant pas d'accélération matérielle, a des restrictions importantes sur la puissance du processeur. Elle a besoin au minimum d'un processeur dual-core à 2.66-GHz muni de 2 GB de RAM.

[31] http://www.microsoft.com/en-us/kinectforwindows/
[32] http://www.xbox.com/fr-FR/

On remarque donc que le kinect n'est que la partie du système qui permet de détecter la profondeur et la transmettre à l'unité de calcul. On va donc s'intéresser aux capteurs de profondeur en général.

Capteurs de profondeur

La profondeur est un moyen de perception de l'environnement qui permet d'obtenir une abstraction géométrique de l'espace. (Howard 2012) décrit différentes façons de percevoir la profondeur à travers la vision. La stéréoscopie étant la technique la plus répandue car elle permet la meilleure approximation comparée aux autres techniques visuelles. Une liste non exhaustive des méthodes de perception visuelle de la profondeur sont :

- La **stéréoscopie** ou stéréovision.
- L'accommodation de la **focale**.
- Représentation **perspective**.
- Le **dégradé** de couleurs « Depth from shading ».
- La **parallaxe** du mouvement.
- Autres informations ou connaissances préalables accompagnant l'image.

Les techniques basées sur la vision sont similaires à la vision humaine. Ils en héritent les lacunes, mais sont par contre applicables facilement en utilisant des caméras classiques. La perception de la profondeur, quant à elle, dépasse le cadre de la vision. On considère que les principales techniques alternatives de perception de la profondeur sont :

- Les systèmes **LIDAR**[33], classiquement utilisés en géologie, ont assuré leur place en robotique et systèmes autonomes avec des miroirs pivotant à grande vitesse permettant la couverture de lignes du champ de vision. Le Velodyne LIDAR[34] avec de multiples faisceaux et une base tournante permet une bonne couverture de l'espace périphérique.
- Les **caméras temps de vol** « Time-of-Flight ToF camera », dont le principe de fonctionnement et les applications sont décrits dans (Lee et al. 2013), représentent une classe de LIDAR, mais avec une capture de toute la scène à chaque impulsion de lumière.
- La **lumière structurée** : on dispose d'une récapitulation sur les lumières structurées invisibles dans (Fofi et al. 2004).

Le Kinect combine la stéréovision et la lumière structurée. Pour le calcul de l'appariement, un ASIC a été développé. Le PS1080, présenté dans la figure 5-3, permet à ce périphérique de décharger le PC du calcul qui aurait été prohibitif et impliquant des ressources de calcul non disponibles pour le grand public. Les ASICs ne deviennent rentables qu'à très grande échelle commerciale, ce qui n'a été possible que grâce à l'industrie des jeux vidéo qui a commandé la construction de ce type de capteurs.

[33] Light Detection And Ranging
[34] http://velodynelidar.com/lidar/lidar.aspx

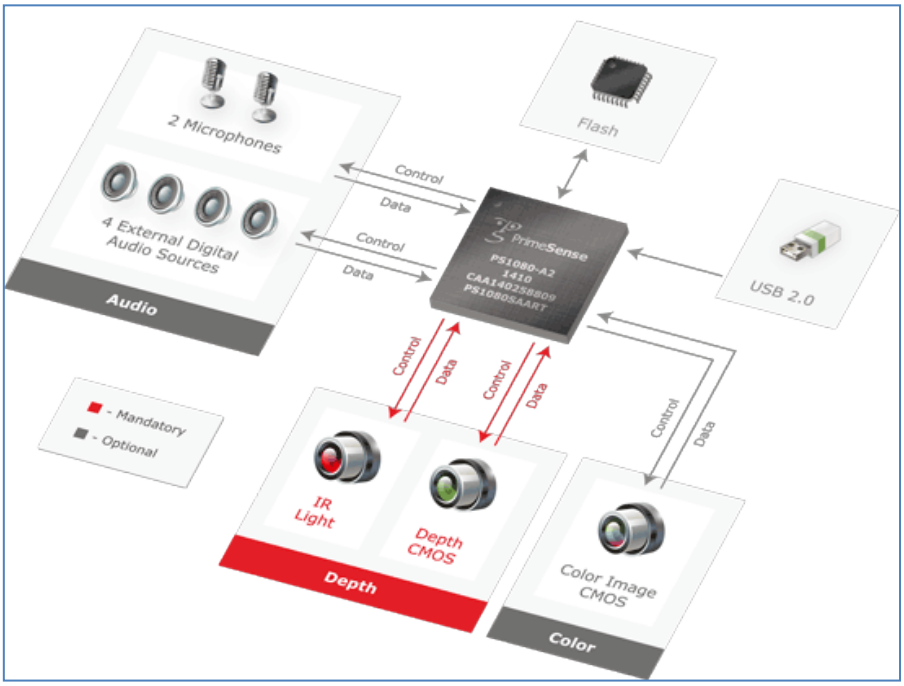

Figure 5-3 : PS1080 (Primesense[35]).

5.2 Modélisation du système avec SysML

Le choix pour la modélisation du système s'est porté sur SysML car l'objectif est de donner une vision globale du projet. Les différents diagrammes permettent d'introduire le système en faisant abstraction des détails de l'intégration.

On a utilisé certaines particularités de SysML. Pour la différence entre UML et SysML, se référer au tutoriel (Friedenthal et al. 2009).

Diagramme de définition des blocs Posture par multiples canaux RGBD

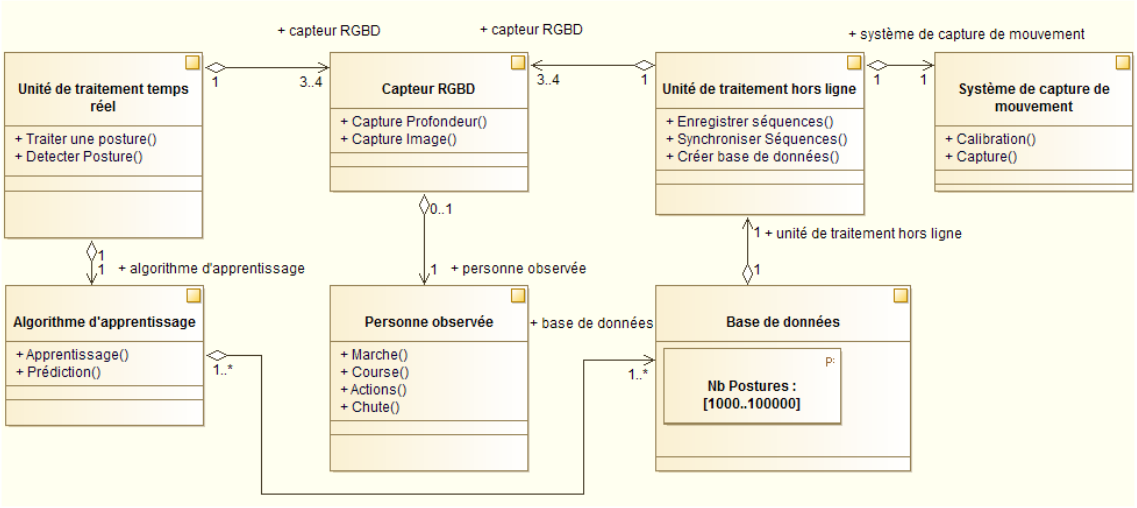

Figure 5-4 : Diagramme de définition des blocs.

[35] http://www.primesense.com/solutions/technology/

La figure 5-4 présente toutes les entités en interaction. Elle contient les éléments nécessaires au traitement hors-ligne pour la construction de la base de données. Cette base de données est utilisée par l'algorithme d'apprentissage. Cet algorithme exécute la classification nécessaire à la reconstruction de la posture lors du traitement en ligne.

La figure 5-5 présente le diagramme d'activité de l'unité de traitement temps réel. C'est l'unité qui nous intéresse dans ce chapitre car elle représente l'unité nécessitant un matériel dédié correspondant au contexte de l'application. On note la présence de lignes continues représentant le flot de données et de lignes discontinues représentant le flot de contrôle. Le flot de contrôle indique par exemple que si dans la première étape de traitement, il n'y a pas de personne détectée, le traitement s'arrête et les autres étapes ne sont pas opérées. On remarque aussi l'utilisation des données de profondeur dans deux étapes successives du traitement, ce qui révèle assez tôt le besoin de stocker ces données pour paralléliser le traitement en pipeline.

Diagramme d'activité Unité de traitement temps réel

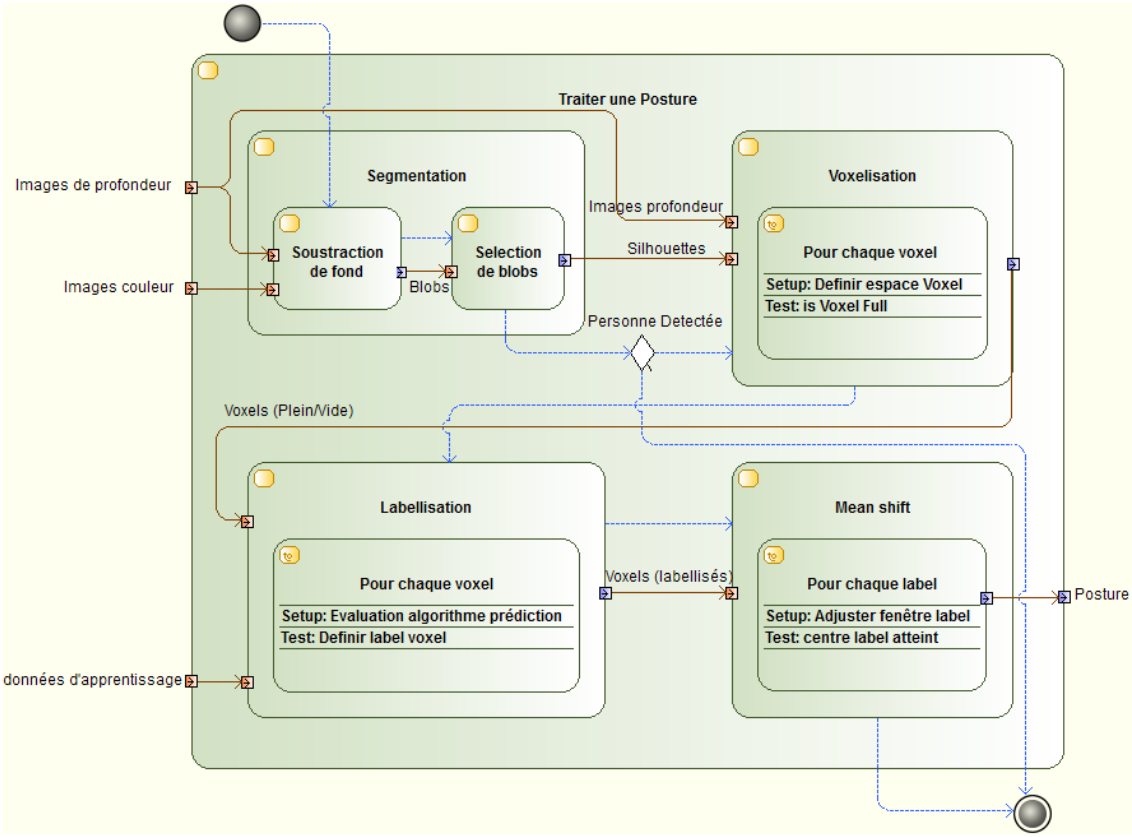

Figure 5-5 : Diagramme d'activité de l'unité de traitement temps réel.

Après avoir présenté le système de façon globale, on va présenter dans la partie suivante, la deuxième étape de notre approche de « rencontre au point milieu ». On va spécifier le bas niveau en définissant les ressources matérielles disponibles et leurs performances.

5.3 Environnement du système embarqué reprogrammable

5.3.1 Introduction

Notre choix du matériel s'est déduit depuis notre répartition des architectures décrite dans la section 5.1.2 ainsi que leur analyse et en considération des critères décrits dans la section 5.1.3.

On vise dans le cadre de cette thèse l'étude de faisabilité du projet, ainsi qu'une étude préliminaire du matériel associé pour un prototype. Le choix s'est porté sur la famille des FPGA, d'abord pour des fins pédagogiques. Car bien que représentant un meilleur choix pour le genre du prototype visé, leur temps d'apprentissage reste considérable. Une expérience est requise ainsi qu'un environnement d'outils déjà préétabli. C'est ce qui permet de tenir les temps impartis du projet. Et c'est justement ce point-là qui a attiré notre attention et qui a fait l'objet de notre contribution dans cette thématique. C'est l'étude de l'environnement et plus spécifiquement, des composants d'un système sur puce programmable « SOPC ». C'est une approche qui permet de contenir les éléments de base dans des boîtes d'un langage de plus haut niveau. Ce langage de haut niveau est concrétisé par un outil de synthèse, dans notre cas, le SOPC Builder[36] renommé en QSys[37] dans les versions récentes du système de développement. Ces outils sont spécifiques au constructeur Altera[38], mais on constate la démocratisation de ces outils de synthèse en haut niveau chez d'autres constructeurs également, comme « Embedded Development Kit » , « Software Developement Kit », « System Generator for DSP » pour Xilinx[39]. On remarque que les FPGA ne sont plus considérés comme des systèmes isolés des autres moyens de développement séquentiels sur processeurs. On profite donc de l'avantage des deux approches : l'accélération matérielle avec des périphériques et le contrôle par un processeur. Cela rend cette puissance de traitement matérielle aussi flexible d'usage et de configuration que les architectures logicielles. Notre contribution consiste à adapter cette méthode à la vision en utilisant des modules de base qui seront explicités par la suite.

5.3.2 Environnement de développement

La figure 5-6.(a) représente un banc stéréo déjà développé dans l'équipe et qui a bénéficié des composants détaillés ultérieurement. Ce banc stéréo opère à une fréquence de 100 Hz avec deux caméras VGA 640x480. L'entrée et la sortie du module de traitement sont de type ***CameraLink***. La figure 5-6.(b) représente un robot mobile équipé d'un système de vision multi caméras.

[36] http://www.altera.com/literature/lit-sop.jsp
[37] http://www.altera.com/products/software/quartus-ii/subscription-edition/qsys/qts-qsys.html
[38] http://www.altera.com/
[39] http://www.xilinx.com/products/design-tools/ise-design-suite/

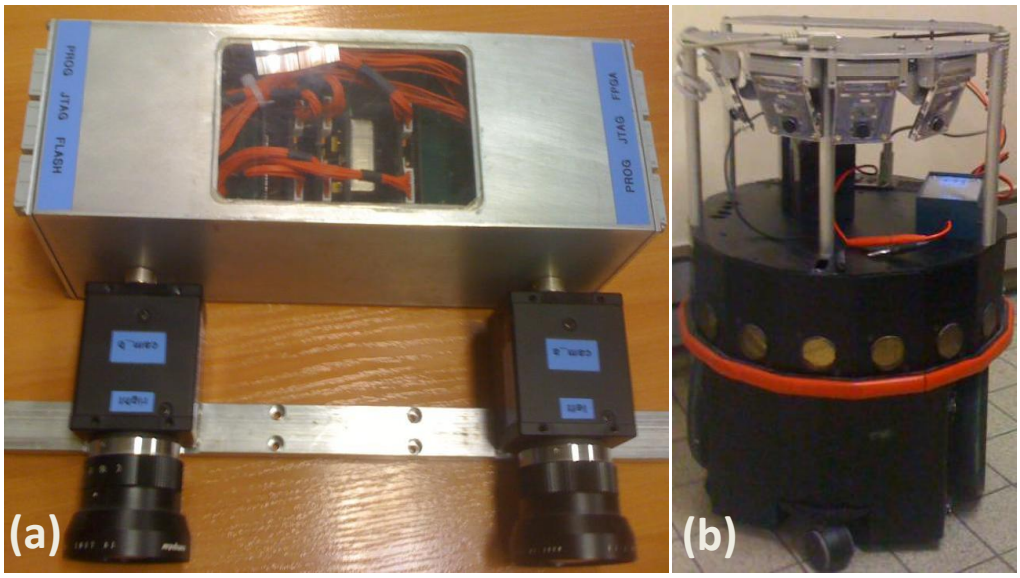

Figure 5-6 : a - Banc Stéréo 100 Hz b – Robot équipé du système de vision (LAAS-CNRS).

Matériel

Nos développements sont généralement menés avec des kits de développement, et dans certains cas avec des prototypes plus avancés comme le cas du banc stéréo développé avec une société partenaire du projet. Le module de traitement du banc stéréo comprend 3 cartes interconnectées basées sur des composants cyclones II.

La validation fonctionnelle est tout d'abord menée sur une carte de développement Terasic[40] de type DE2-70 présentée dans la section 5.5.

L'intégration de multiples fonctionnalités nécessitant plus de ressources est réalisée sur un kit de développement à base du composant StratixIII. Le kit est le centre du système de vision multi caméras représenté par la figure 5-7. C'est ce système qui a été équipé sur le robot indiqué dans la figure 5-6.(b). Le composant StratixIII (référence EP3SL150F1152) est muni de 140K éléments logiques et de 384 multiplieurs 18x18. Les mémoires disponibles sont :

- Une PSRAM à doubles modules délivrant une interface 32 bits avec un mode de fonctionnement synchrone à 100 MHz.
- Deux modules DDR2 indépendants de 8 bits et de 16 MO chacun, opérant à 200 MHz, délivrant chacun un débit de 100 Méga mots de 32 bits par seconde.
- Un module DDR2-DIMM de 72 bits avec 1GO et opérant à 400 MHz.
- Une mémoire QDRII+ de 72Mbit opérant à double port pour lecture et écriture avec des mots de 18bit.

[40] http://www.terasic.com

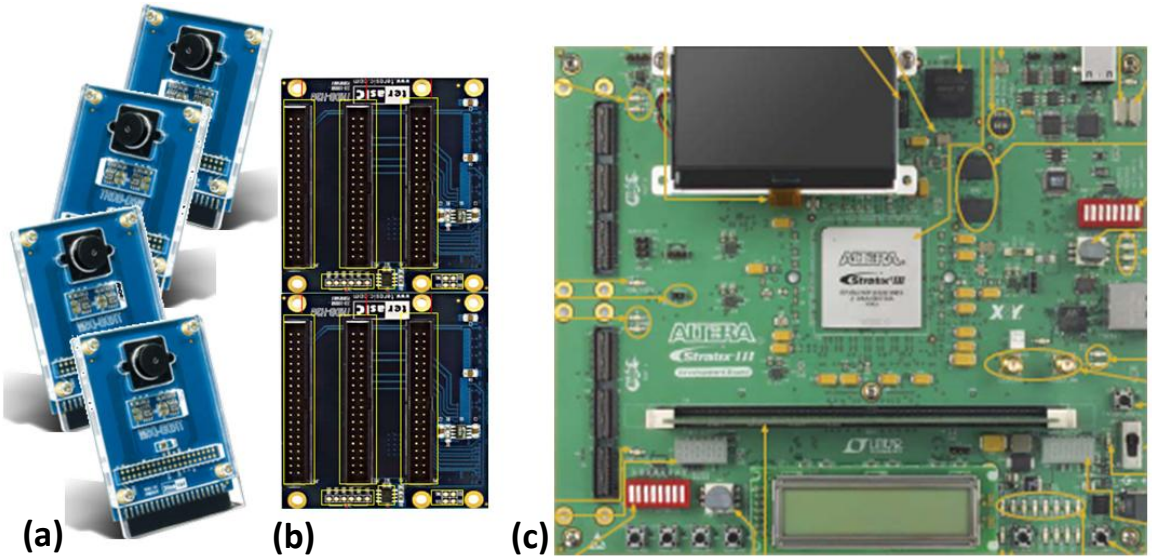

Figure 5-7 : Plate-forme de l'application multi caméras – a – caméras – b – adaptateurs – c – Kit de développement Stratix III (TerasicD5M[41], AlteraDevKit[42]).

Les caméras utilisées ici sont des D5M de Terasic. Ce sont des caméras avec des sorties parallèles. Jusqu'à 4 caméras peuvent être connectées sur le kit de développement avec des adaptateurs.

Outils de conception

On a utilisé les outils proposés par Altera, le constructeur des composants. Ce choix a été dicté par le partenaire industriel qui comptait développer une nouvelle version des prototypes dont les versions précédentes ont déjà été développées avec des composants Altera. Si ce choix a été hérité plutôt que décidé, il nous a semblé nécessaire de pouvoir migrer nos projets d'une famille de composants à une autre provenant d'un constructeur différent. Les deux principaux constructeurs sont Altera et Xilinx[43], mais d'autres existent pour des applications plus spécifiques. Altera nous a semblé plus adapté car il y a plus de documentation et d'exemples de prise en main disponibles. Alors que Xilinx est plus adapté pour les professionnels et expérimentés, avec plus de composants à la pointe de la technologie, mais sans la facilité de prise en main nécessaire pour les débutants.

Le principal outil de synthèse d'Altera est l'environnement « QuartusII » qui à son tour interagit avec d'autres logiciels, dans notre cas, « SOPC Builder » ou « QSys ». Cet outil permet l'interconnexion graphique de modules interfacés via un bus propriétaire : le bus « Avalon[44] ». Le bus « Avalon » est un bus d'interconnexion dont les paramètres exacts restent flexibles et dépendent de l'implémentation. Les deux principales variantes sont le mode d'adressage mémoire et le mode de transmission de paquets en série.

L'outil « SOPC Builder » est disponible avec une bibliothèque initiale. Notre approche a été d'enrichir cette bibliothèque avec des composants facilitant l'interconnexion à la mémoire et d'autres spécifiques au traitement d'images.

Cet outil peut être utilisé sans inclure de processeur au système conçu, mais sa logique a été étudiée pour faciliter l'intégration d'un processeur au cœur du système et rendre instantanée

[41] http://www.terasic.com.tw/cgi-bin/page/archive.pl?&No=281
[42] http://www.altera.com/products/devkits/altera/kit-siii-host.html
[43] http://www.xilinx.com
[44] http://www.altera.com/literature/manual/mnl_avalon_spec.pdf

son interconnexion avec les modules périphériques. Le système gère l'affectation des espaces d'adresse parmi d'autres détails d'implémentation. Le processeur que l'on a utilisé est le « NIOSII[45] », c'est un « soft core ». Il est inclus dans l'architecture qui sera synthétisée et routée sur le circuit FPGA et il peut être synthétisé en différentes versions qui ont différents compromis entre performances et ressources utilisées. Les premières couches logicielles sont générées automatiquement par le système. Le logiciel « NIOS-IDE » permet la compilation et la programmation du code. « NIOS-IDE » a été conçu à partir d'Eclipse. On rappelle que le processeur intégré dans le système n'a pas vocation, dans notre application, à calculer lui-même des fonctions de traitement d'images, mais il doit jouer un rôle de coordination et de configuration.

5.3.3 Composants de base

Dans cette section, on va présenter deux types de composants : des composants facilitant l'interconnexion et la conversion d'un type de flux à un autre, et d'autres spécifiques aux fonctions de traitement d'images.

Standard Avalon et sa spécification aux images

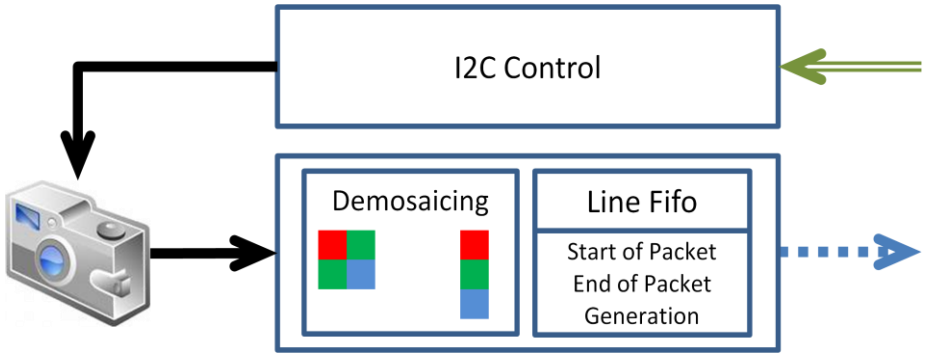

Figure 5-8 : Caméra vers Avalon.

Nous avons décidé d'adopter un format d'image VGA (640x480). Nous avons utilisé 32 bits par pixel composé du rouge, vert, bleu et un canal de transparence. Chaque paquet Avalon contient une image entière. La taille de l'image est toujours la même durant tout le processus de traitement.

Les caméras ont un fonctionnement spécifique selon la famille du capteur utilisé. Il est judicieux de faire abstraction à ces signaux en les transformant en un signal standard permettant une intégration plus facile avec le reste des modules Avalon. Pour cela, deux modules sont nécessaires comme le montre la figure 5-8. Le module de contrôle I2C permet de configurer la caméra directement avec le processeur NIOSII. La configuration de la caméra, temps d'exposition et autres paramètres sont donc configurables par logiciel. Le module de conversion du format de l'image applique l'opération de démosaïquage qui consiste à convertir le format RGGB à notre format XRGB.

[45] http://www.altera.com/devices/processor/nios2/ni2-index.html

Composants d'écriture et de lecture en mémoire

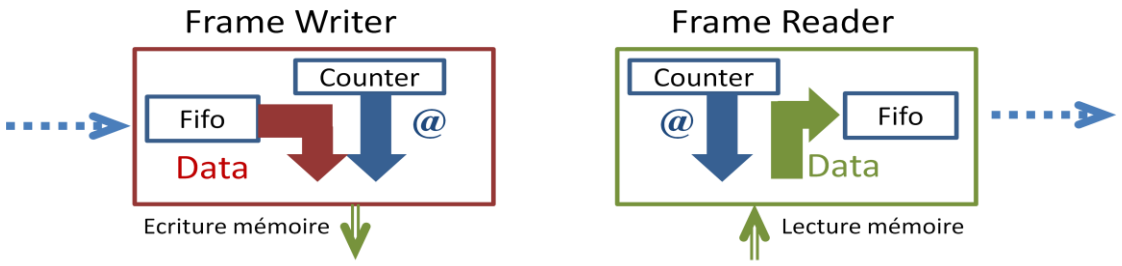

Figure 5-9 : Transformation des flux.

Les composants représentés par la figure 5-9 permettent d'écrire et de lire les paquets d'images ou autres dans et à partir de la mémoire.

Quand on a un paquet de données qui se transfère en série et que l'on veut l'écrire dans une plage d'adresses spécifiques de la mémoire, il suffit de définir l'adresse de départ où se trouvera le premier pixel, et un compteur se charge d'incrémenter l'adresse au fur et à mesure de l'arrivée des autres pixels. Ces adresses, où vont se loger les images, peuvent être configurées par logiciel. L'ensemble de ces deux composants constitue un « Frame Buffer ». L'objectif de les avoir séparés en deux modules indépendants, permet plus de configurations de fonctionnement. On peut avoir un nombre de modules de lecture inégal au nombre de modules d'écriture.

Les FIFOs (Premier arrivé premier servi) permettent de connecter plusieurs modules à la même mémoire. Sous réserve que la somme des flux ne dépasse pas la bande passante commune, l'accès reste garanti. L'arbitrage de tous les modules est réparti, car l'accès mémoire n'est réservé que durant le temps nécessaire pour vider le FIFO.

Module de transformation

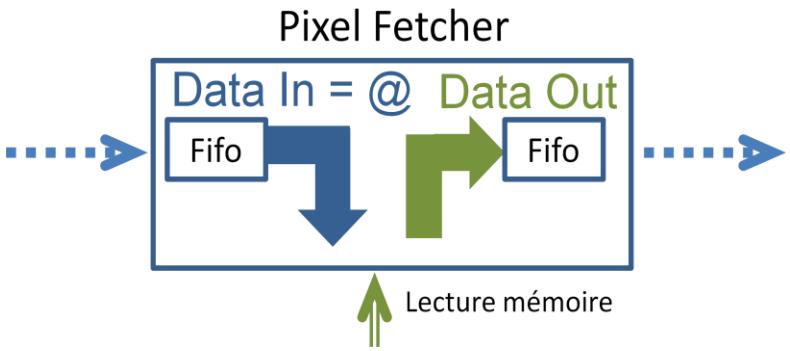

Figure 5-10 : Récupération des pixels.

La figure 5-10 représente le module nécessaire à toutes les transformations d'images. C'est le récupérateur de pixels. Il est issu d'une transformation du « Frame Reader ». Il n'a pas un compteur qui incrémente les adresses à lire de façon monotone, mais il a en entrée un flux Avalon qui indique à chaque nouveau pixel, quelle sera la nouvelle adresse à charger. Les adresses à charger dépendent du contenu en entrée et ne sont pas nécessairement continues. Cela peut entraîner une baisse de la bande passante selon le type de mémoire. D'après les tests effectués avec notre équipement, une mémoire DDR est 5 à 10 fois plus lente avec un accès aléatoire qu'avec un accès séquentiel.

Le flux des adresses à charger peut provenir de deux sources différentes : de la mémoire, avec l'utilisation d'un « Frame Reader » en amont, ou calculé de façon dynamique avec un composant

spécifique. Le tableau 5-2 compare ces deux techniques pour renseigner dans quel cas l'une est plus appropriée que l'autre.

Transformation avec table en mémoire	Transformation dynamique
Statique, ou le temps d'écriture pour la mise à jour	Dynamique
Même module matériel réutilisable	Conception spécifique à chaque transformation
Ressources mémoires pour la table	Ressources FPGA, multiplicateurs, registres
Consommation de bande passante de lecture	Pas besoin d'accès mémoire
Pas de limite de complexité	Complexité croissante avec la complexité de la fonction de transformation
Accès à de multiples images (fusion d'images)	Accès à une image unique
Possibilité de concaténer les tables de transformations en une seule	La concaténation doit concaténer les blocs avec perte de résolution

Tableau 5-2 : Comparaison des transformations.

Correction de distorsion

La formule de correction de distorsion dépend de la lentille utilisée. On utilise un modèle mathématique standard (Brown 1971), pour lequel la procédure d'étalonnage permet de déterminer les paramètres.

Plusieurs types de distorsions sont possibles pour une lentille. La distorsion radiale, dont la formule de correction est la suivante :

$$x_{corrigée} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{corrigée} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Et une distorsion tangentielle, qu'on n'a pas eu à appliquer mais que l'on cite à titre de référence :

$$x_{corrigée} = x + [2p_1 y + p_2(r^2 + 2x^2)]$$
$$y_{corrigée} = y + [p_1(r^2 + 2y^2) + 2p_2 x]$$

La figure 5-11 illustre le principe de fonctionnement du module.

On définit ***A*** comme la coordonnée corrigée du point et ***B*** la coordonnée de notre point de référence. Quand le flux d'image a besoin du point ***A***, le module de correction applique la transformée et demande plutôt le point corrigé ***B***. De la sorte, l'image en sortie est rectifiée.

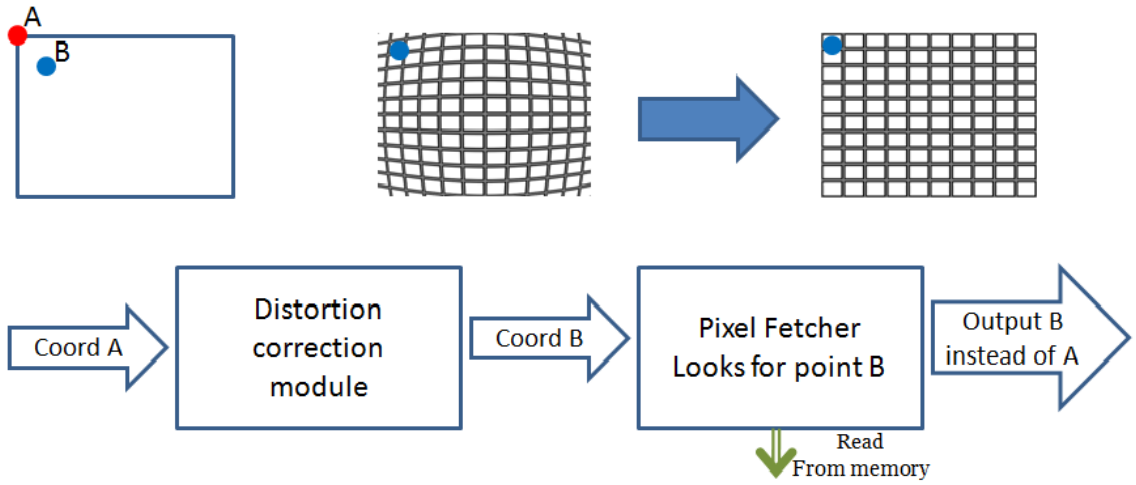

Figure 5-11 : principe de fonctionnement de la correction de distorsion.

Homographie

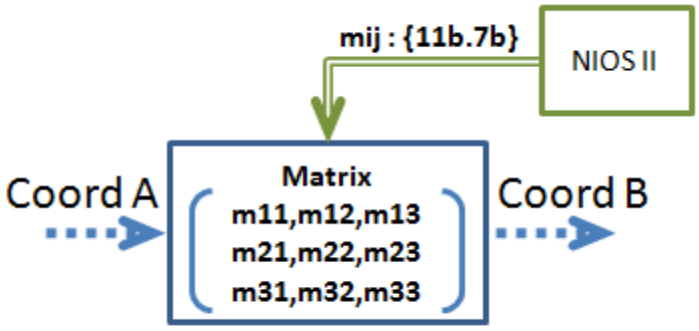

Figure 5-12 : Transformation homographique.

La figure 5-12 représente l'implémentation du module de transformation homographique, dont voici la formulation :

À un pixel en entrée à la coordonnée (x_e, y_e) on fait correspondre une nouvelle coordonnée (x_s, y_s) de sorte que :

$$\begin{pmatrix} sX_e \\ sY_e \\ s \end{pmatrix} = H \begin{pmatrix} X_s \\ Y_s \\ 1 \end{pmatrix}$$

Avec H étant une matrice 3x3 calculée en ligne par le processeur NIOSII en fonction des paramètres de l'application.

Remarquons qu'en fonction de la transformation appliquée, il faut bien synchroniser les tâches de mise à jour de l'image source et de l'application de la transformation. Au pire des cas, le premier pixel à sortir peut être le dernier pixel mis à jour dans l'image source.

5.3.4 Paramètres influençant la performance

Il est judicieux de considérer certains détails comme ceux que nous allons présenter qui ont un impact sur l'optimisation de l'architecture.

L'indépendance de l'interface mémoire

L'outil de développement SOPC Builder facilite la gestion d'une interface mémoire en gérant automatiquement l'adaptation de la largeur des mots. Cela doit néanmoins rentrer dans le cadre de l'étude de chaque projet, car si l'on a besoin de mots dont la largeur est différente de la largeur réelle de la mémoire, l'outil va automatiquement adapter cela, mais au prix de multiples lectures et multiples accès mémoire.

Le partage de la bande passante

SOPC Builder permet à plusieurs maîtres d'accéder au même esclave. Un composant « Frame Reader » ou « Frame Writer » a un accès au bus en mode maître tandis que la mémoire a un accès en mode esclave. Nous avons élaboré un arbitrage indépendant déporté dans chaque composant maître. Le composant a un FIFO, quand il a besoin de le remplir, il récupère l'accès au bus, suite à son premier accès, il réserve le bus en sa faveur. Une fois le FIFO rempli, il libère l'accès au bus afin que d'autres composants puissent accéder pour remplir leur FIFO. Ce mode de fonctionnement dépend d'un équilibre délicat qui définit quand chaque composant doit réserver le bus et quand le libérer. Dans la nouvelle version dénommée « QSys », il y a eu un changement dans la spécification du bus Avalon qui nous a amené à adapter les composants. En même temps, de nouvelles options permettent de configurer directement le mode « Burst » pendant l'accès au bus. Cette configuration permet de nous affranchir de notre mode d'arbitrage qui est maintenant régi par le système « QSys ». Chaque composant accède de façon transparente au bus, et l'accès ne lui est dédié que pendant un nombre de transactions bien défini.

Ces deux modes de fonctionnement similaires ont pour objectif de lire un certain nombre de mots à chaque accès (fixés à 16, 32 ou 64 pendant nos expérimentations). Cette lecture en mode « burst » optimise l'accès à la mémoire dans le cas où tous les esclaves lisent des zones séquentielles de la mémoire. Si chaque esclave ne lisait qu'un seul mot pendant son accès, l'accès global de tous les esclaves à la mémoire serait totalement aléatoire et ne bénéficierait pas de la performance optimale de la mémoire. On a relevé durant nos essais que l'accès aléatoire prend 8 à 10 coups d'horloge alors que l'accès séquentiel n'en prend qu'un seul.

5.3.5 Exemples de pipelines et leurs performances

Le tableau 5-3 indique les bandes passantes mémoire de différents composants.

	Paramètres	Bande passante
1x flux caméra	30 Hz x 640x480 pixels x 32 bits	9.2 Méga mot /s
Accès mémoire DDR2	200 MHz x 2 (DDR) x 8 bits	100 Méga mot /s
1x Pixel Fetcher accès DDR	10 Méga accès par seconde	90 Méga mot /s
1x Pixel Fetcher accès SRAM	10 Méga accès par seconde	10 Méga mot /s

Tableau 5-3 : Paramètre des flux et bande passante associée.

Cela nous permet d'allouer les mémoires en fonction des fonctionnalités.

Accès mémoire multi caméras

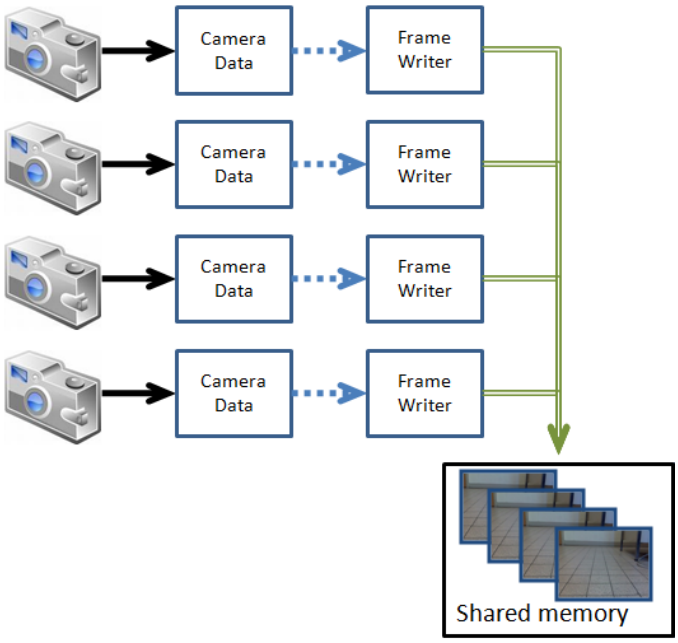

Figure 5-13 : Accès multi caméras.

Plusieurs flux de caméras peuvent être connectés à la même mémoire partagée en utilisant le composant « Frame Writer ». Le flux des caméras est continu alors que l'accès à la mémoire est alterné entre chaque « Frame Writer ». Ce sont les FIFO intégrées dans ces composants qui permettent d'emmagasiner le flux en attendant de gagner l'accès à la mémoire. Le pourcentage de temps perdu en alternant l'accès de la mémoire entre « Frame Writer » a été relevé à 5 % pour 4 accès, ce qui reste raisonnable tant qu'on n'avoisine pas la limite de la bande passante.

Concaténation de modules de traitement

La figure 5-14 illustre la concaténation d'une correction de distorsion avec une transformation homographique. La transformation homographique a des paramètres variables. Dans l'application du robot mobile (figure 5-6 (b)), c'est l'orientation courante du robot qui met à jour dynamiquement ces paramètres.

Dans le cas d'utilisation d'une SRAM, la transformation s'exécute à la fréquence de l'horloge de 100 MHz et pour les 307 200 pixels. Cela ne dure que 3 ms.

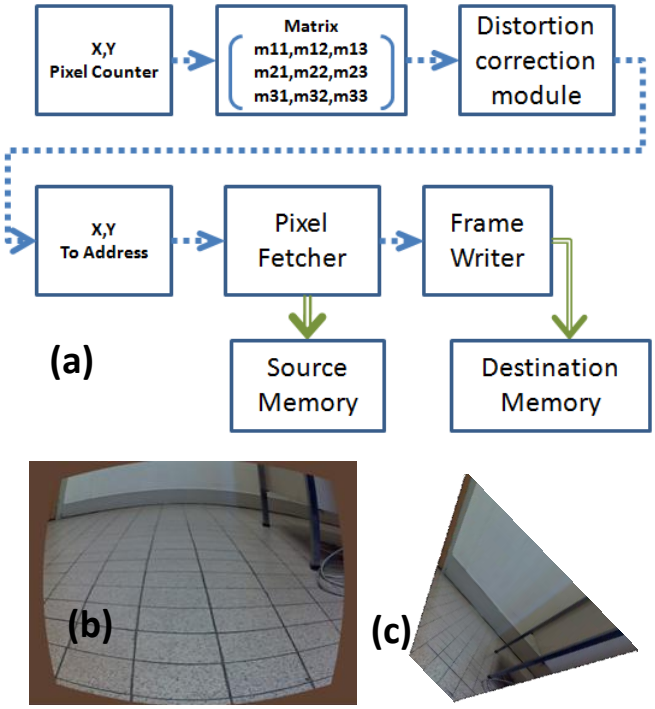

Figure 5-14 : Concaténation de modules de traitement – (a) connexion des modules – (b) Image source distordue – (c) Image rectifiée et transformée.

5.4 Placement des ressources avec les solutions retenues

Nous avons retenu deux solutions. Une solution PC pour le développement et l'évaluation ainsi que pour les applications en laboratoire, et une solution FPGA pour des systèmes embarqués, à déployer dans des environnements contraints. Ces solutions sont confrontées à la solution existante de console de jeux dont le contexte est ludique.

Le tableau 5-4 présente le placement des mêmes fonctionnalités dans des ressources matérielles différentes selon le contexte applicatif.

La console utilise un capteur kinect. On a noté par une croix la ressource affectée. Les cases grisées représentent une fonction non applicable dans le cas d'une console vu que notre solution offre plus de perspectives en utilisant plusieurs points de vue provenant de plusieurs capteurs.

La voxellisation et la labellisation dans le contexte applicatif relatif au PC, peuvent optionnellement être implémentées sur CPU ou sur GPU (représenté par ***o*** sur le tableau). Leur implémentation sur GPU est justifiée par leur complexité de branchement réduite et leur nombre d'instructions élevé.

Solution	Ressource \ Fonction	Capture	Calcul profondeur	Soustraction de fond	Sélection des Blobs	Voxelisation	Labellisation	Mean shift
Console	Kinect – Capteur	x						
	Kinect – PS1080		x	x	x			
	Console – Processeur							x
	Console – GPU						x	
PC	Xtion – Capteur	x						
	Xtion – PS1080		x	o				
	Processeur			o	x	o	o	x
	GPU			o		o	o	
FPGA	Capteur externe	x						
	Module spécifique		x	x		x	x	
	Soft-core				x			x

Tableau 5-4 : Placement des fonctions.

Par la suite, on va donner plus de précisions sur les solutions retenues : PC, GPU et FPGA.

On a consacré une partie indépendante à l'architecture GPU, car elle représente une différence majeure comparée à l'intégration sur le processeur central du PC. Cette architecture GPU se retrouve notamment dans la solution existante sur console et aussi optionnellement dans la solution sur PC.

5.4.1 Architecture multi xtion sur PC

Figure 5-15 : Architecture PCArchitecture PC multi Capteurs (PC HP[46]-XtionProLive[47]).

Cette solution est la plus facile à mettre en œuvre, grâce aux capteurs disponibles dans le commerce. Ces capteurs ont une interface USB, pour laquelle un pilote spécifique est nécessaire. Cela rend très complexe leur utilisation avec un processeur embarqué non compatible PC. La

[46] http://h20386.www2.hp.com/FranceStore/SubCategories.aspx?pid=C326900
[47] http://www.asus.com/Multimedia/Xtion_PRO_LIVE/

machine utilisée est un serveur HP Z800 avec deux processeurs de 4 coeurs chacun. Même dans une configuration PC, un bus USB est nécessaire pour chaque capteur (il n'est pas possible de connecter 2 capteurs sur deux connecteurs USB qui à l'intérieur du PC partagent le même bus), de plus les ports en version USB3 ne sont pas utilisables avec ces capteurs en version USB2.

Vu que ces capteurs ont été conçus pour un fonctionnement en module unitaire, en cas de support de plusieurs capteurs, leur fonctionnement est totalement indépendant. À titre d'exemple, l'inconvénient majeur de cette solution est qu'il n'est pas possible de synchroniser la capture des différents capteurs avec un signal de trigger unique. Même s'il est préférable de concevoir un système synchrone, ce léger déphasage induit une erreur qui n'impacte pas la capture et la fusion des données.

Un autre inconvénient vient de l'interférence entre capteurs, qui est considérée minime sur les surfaces du corps d'une personne. Vu la forme plutôt convexe du corps, chaque zone est principalement illuminée et visible par un seul capteur à la fois, ou par deux, avec un niveau de bruitage acceptable comme le montrent les séquences capturées.

5.4.2 Présentation de l'architecture GPU

On considère le GPU comme un accélérateur d'exécution qui peut éventuellement équiper un environnement PC comme celui présenté dans la section précédente.

La figure 5-16 nous donne des détails sur les spécificités de l'architecture d'un GPU. Cette organisation reste la même d'un GPU à un autre (hormis les anciennes générations de GPU spécifiquement graphiques). Ce qui varie d'un GPU à un autre, c'est :

- Le nombre de ***blocs*** par ***grille***, le nombre de « ***thread »*** par ***bloc***.
- L'indépendance d'exécution et le parallélisme du lancement de tâches par le PC ***hôte***.
- Les tailles des mémoires locales et spécifiques, ainsi que l'accès à ces mémoires.
- La bande passante entre le GPU et sa mémoire, ainsi que la bande passante du transfert entre le PC ***hôte*** et la mémoire GPU.

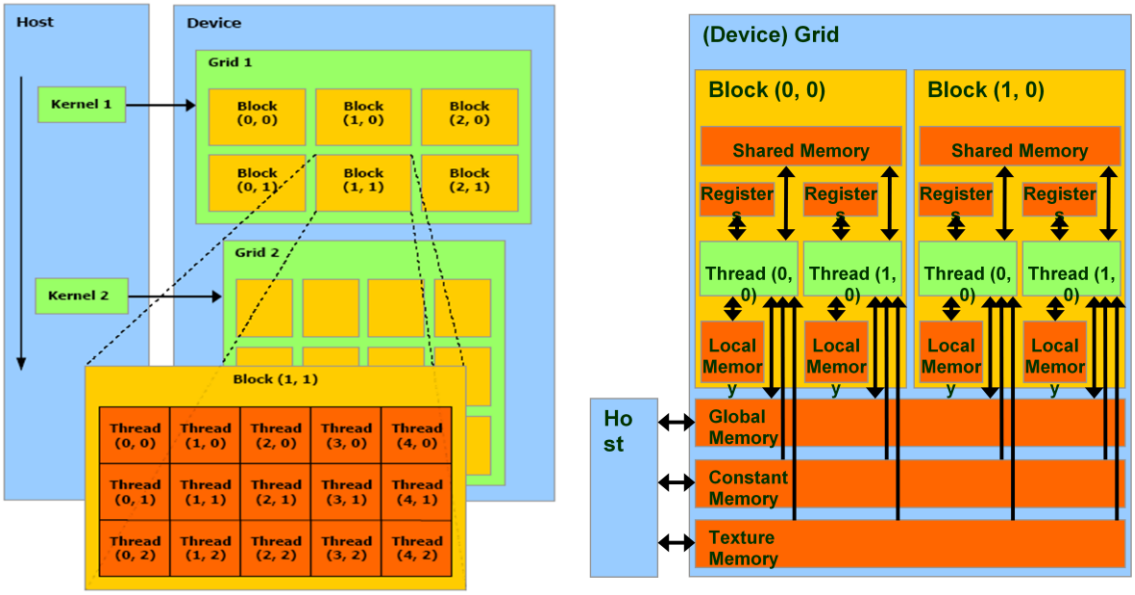

Figure 5-16 : Architecture d'un GPU : (gauche) Exécution – (droite) Mémoires (Nvidia[48]).

On réfère au programme lancé sur le GPU, selon le contexte, par : tâche, programme GPU, noyaux ou en anglais « Kernel ». Le GPU a besoin du PC pour lancer le pilote de la carte graphique. C'est le pilote exécuté sur PC qui prépare le lancement du ***noyau***. Cela produit un temps de latence « CPU Overhead ».

Chaque ***grille*** n'exécute que le même ***noyau***. Les différents ***blocs*** peuvent être à des endroits différents du même ***noyau***, mais des groupes de « thread » du même ***bloc*** partagent le même contrôleur d'instruction. Cela augmente l'efficacité des GPU concernant le nombre de transistors par instruction exécutée, mais le limite à n'exécuter qu'un seul « thread » par groupe. Ce qui se traduit par une efficacité croissante de l'usage du GPU pour les programmes largement similaires entre les nombreuses données concernées (exemple, même traitement pour tous les pixels d'une image).

Le tableau 5-5 indique les paramètres de la carte graphique mise à notre disposition pour le test présenté dans la section 5.4.4.

Référence	Quadro FX 4800
Nombre de cœurs	192
Mémoire	1.5 GO
Interface de la mémoire	384 bit
Bande passante de la mémoire	76.8 GO/s
Puissance Maximale	150 W

Tableau 5-5 : Caractéristiques de la carte graphique utilisée.

[48] www.nvidia.com

5.4.3 Architecture smart caméras multi vues sur FPGA

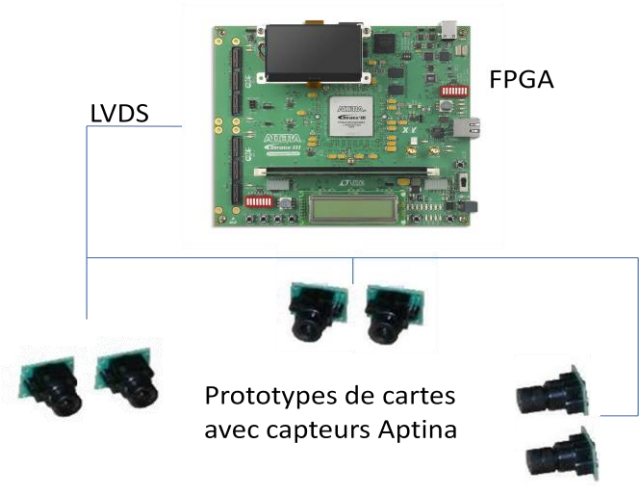

Figure 5-17 : Architecture embarquée sur FPGA (AlteraDevKit[49]).

Cette architecture a été conçue pour une application totalement embarquée qui inclut la génération des cartes de disparité (profondeur, comme c'est le cas dans les capteurs xtions), ainsi que l'intégration des fonctions de perception de posture.

Le choix pour générer la profondeur s'est porté sur la stéréovision classique. Ce choix est motivé par son intégration sur FPGA lors de précédents projets mais aussi par le problème d'interférence que suscite l'utilisation de plusieurs kinects ou xtions, dirigés sur la même cible. La vision active est en effet un inconvénient dans le cadre multi vue. Comme la stéréovision passive peut aussi profiter d'une lumière active qui enrichit les surfaces en motifs différentiables, il est possible dans notre cas d'utiliser une lumière unique pour toute la scène, ou plusieurs qui soient synchronisées avec les temps de capture de chacun des capteurs.

5.4.4 Comparaison d'une fonction sur CPU, FPGA, GPU

État de l'art sur la comparaison des fonctions sur CPU, FPGA, GPU

Toute tentative de comparaison objective des différentes plates-formes CPU, FPGA, GPU, est une réponse à un questionnement légitime, mais qui risque de transformer la question en un piège si l'on compare un paramètre unique sans considérer la totalité du système ni le besoin de l'application finale.

En fonction de l'origine de l'initiative de comparaison, on risque d'avoir une opinion subjective. Même si elle n'est pas intentionnelle, on risque de développer mieux la solution qui fait partie de son domaine d'expertise et de formuler des hypothèses trop optimistes. Le plus judicieux serait de se limiter à des conclusions liées à son propre cadre applicatif et expérimental.

On peut citer à titre d'exemple (Gac et al. 2008) qui considère l'intégration d'une application de tomographie 3D. La conclusion est que le GPU apporte les meilleures performances, mais un FPGA tire un meilleur profit de l'architecture en fonction des coups d'horloge. Une solution en ASIC (que nous avons écarté vu que cela dépasse notre cadre applicatif), avec les mêmes ressources qu'un GPU **serait** (verbe conjugué au conditionnel) 2 fois plus rapide qu'un GPU et 20 fois plus rapide qu'un CPU.

[49] [49] http://www.altera.com/products/devkits/altera/kit-siii-host.html

(Kalarot & Morris 2010) considère l'application de stéréovision et déclare que : **au moins pour l'application étudiée,** le FPGA a de meilleures performances. L'avantage du FPGA est la possibilité de concaténer les fonctions (tel que nous l'avions remarqué dans la section 5.3.5). L'inconvénient majeur du GPU est la latence d'exécution qui accompagne le lancement de chaque noyau et le temps de transfert des données entre la mémoire système et la mémoire GPU reste négligeable. Le GPU utilisé est un « GTX 280 » et le composant FPGA utilisé est un « Stratix III ». Nous remarquons que les GPU ne disposent pas tous de la même performance. Elle varie selon leur bande passante, nombre de cœurs, et autres options de l'architecture.

(Chase et al. 2008) se prête à la comparaison du flot optique sur FPGA et GPU. Une remarque intéressante est que le temps de développement sur FPGA a été dix fois plus important que le temps de développement sur GPU. Ils indiquent clairement que l'algorithme a d'abord été étudié et développé sur FPGA, ce qui représente le but de l'application. Le GPU n'a été abordé qu'à titre comparatif.

La conclusion qui nous a semblé la plus appropriée est que certaines fonctions sont plus rapides sur FPGA et d'autres sur GPU. L'intérêt d'avoir développé les deux versions est de dresser une liste des détails architecturaux qui font que chaque point avantage soit le FPGA soit le GPU (Flexibilité, entrées sorties configurables, calcul par bloc, pipelining, temps de développement en fonction du niveau de maîtrise).

Le constructeur Altera a utilisé comme modèle de test ou « benchmark » l'application de Monte Carlo « Black-Scholes » présentée dans ce papier « white paper[50] ». L'implémentation a été effectuée avec des noyaux OpenCL, dont l'exécution peut se dérouler sur CPU ou GPU, et aussi sur FPGA avec le compilateur spécifique d'Altera. Les performances sont illustrées dans le tableau 5-6 qui conclut que pour ce genre d'application, Le GPU et FPGA sont plus performants que le CPU, et à performance égale, le composant FPGA a une consommation d'un ordre de grandeur moins importante.

OpenCL Monte Carlo Black-Scholes	Quad Core Xeon	GPU comparable	Stratix IV 530 FPGA
Simulations par seconde	240M	2100M	2200M
Consommation (Watts)	130	215	21

Tableau 5-6 : Résultats de comparaison OpenCL sur CPU-GPU-FPGA (Altera- wp-01173-opencl).

Notre approche a été tout simplement de développer une fonctionnalité unique sur les trois différentes plates-formes, et en tirer quelques remarques qui peuvent justifier et orienter nos futurs choix d'intégration.

Notre comparaison avec l'intégration de la soustraction de fond

L'application de détection de posture a d'abord été développée sur PC qui est un environnement maîtrisé comparativement aux autres. La perspective de ce projet est l'intégration d'un système embarqué sur FPGA. Néanmoins, il reste légitime de se demander pourquoi ne pas avoir opté pour une solution accompagnée de GPU. Il nous a semblé donc judicieux de s'investir pour intégrer une fonction sur GPU, qui permettrait d'avoir une opinion sur l'environnement de développement GPU.

On s'intéresse ici à la soustraction de fond qui est une étape clef précédant la fusion des données dans le processus de détection de postures.

[50] http://www.altera.com/literature/wp/wp-01173-opencl.pdf

Configuration	Temps	Détails
PC	10 ms à 30 ms	Conditionné au nombre de pixels à traiter
GPU	1 à 2 ms	4 ms pour 4 canaux
FPGA	3 ms	Le temps de lire l'image dans la mémoire peut se concaténer avec d'autres fonctions.

Tableau 5-7 : Temps d'exécution sur multiples plates-formes.

Le tableau 5-7 indique les temps d'exécutions sur différentes plates-formes. On remarque que le GPU a les meilleures performances vu qu'il bénéficie du parallélisme des unités de calcul ainsi que d'une bande passante mémoire bien supérieure à celle du Kit de développement qui a été utilisé pour le test.

Le tableau 5-8 détaille les opérations GPU ainsi que le temps nécessaire au CPU pour lancer le noyau de traitement du GPU « CPU Overhead ».

Opération	Temps GPU	CPU Overhead
Copie *Host* à *Device* d'une image	**198 us**	**257 us**
Fusion des images en canaux	**268 us**	**86 us**
Traitement de 4 Images (4 Canaux)	**204 us**	**74 us**

Tableau 5-8 : Détails des opérations GPU.

Le temps de développement peut être subdivisé en temps nécessaire pour maîtriser l'environnement de développement et en temps de développement de fonctions unitaires. Cela diffère énormément en fonction des connaissances préalablement acquises par chaque personne. Même indépendamment de cela, le temps de développement d'une fonction sur FPGA peut durer de deux semaines à plusieurs mois, en fonction de sa flexibilité et de sa paramétrisation. On rajoute à cela des étapes de conception, modélisation, intégration, simulation, test et validation. Pour le GPU, une personne initiée à la programmation sur PC peut avoir une initiation en un mois, au bout duquel, il devient possible d'éviter les pièges et de profiter de différentes techniques d'accélérations. Sans cette initiation, il est possible de développer sur GPU, mais l'approche naïve ne profite pas de l'architecture. Il est aussi utile de profiter des différents moyens de débogages à disposition et de la facilité de lancer les tests et tirer les bilans. Dans ces conditions, une fonctionnalité comme la détection de l'arrière-plan ne nécessiterait que quelques jours.

En comparaison à cela, sur PC, on dispose d'une multitude de librairies comme OpenCV[51], permettant d'avoir la fonction prête à l'utilisation et ceci pour la plupart des algorithmes de la littérature. Le besoin devient donc un besoin d'intégration des fonctionnalités et non pas de leur développement. Par contre, pour des fonctions où l'optimisation est importante, il faut redévelopper avec une méthodologie qui facilite l'intégration avec le reste du programme.

Notre conclusion est que le choix de l'accélération GPU est bien justifié lorsqu'un GPU est disponible. Notre choix de l'environnement FPGA, au lieu du PC, n'est pas motivé par l'accélération pour la réduction du temps d'exécution. Il a été choisi pour un but d'embarquabilité, de compacité des modules matériels utilisés et pour la minimisation de leur consommation. Il est également parfaitement adapté pour un concept de caméras en réseau.

[51] http://opencv.willowgarage.com/wiki/

5.5 Détails de l'implémentation de la fonction soustraction de fond sur FPGA

5.5.1 Formalisation de la soustraction de fond

La soustraction de fond est une problématique largement couverte par la littérature, dont on se contente de citer une revue (Piccardi 2004). Nous avons opté pour la technique Sigma-Delta qui est une méthode efficace et dont le calcul est léger (Manzanera & Richefeu 2004).

Les formules définissant le calcul sont :

$$B_t = \begin{cases} B_{t-1} + 1, & si\left((I_t > B_{t-1}) \wedge \left((D_{t-1} = 0)\right)\right) \\ B_{t-1} - 1, & si\left((I_t < B_{t-1}) \wedge \left((D_{t-1} = 0)\right)\right) \\ B_{t-1} & sinon \end{cases}$$

$$V_t = \begin{cases} V_{t-1} + 1 & si(V_{t-1} < N \times |I_t - B_t|) \\ V_{t-1} - 1 & si(V_{t-1} > N \times |I_t - B_t|) \\ V_{t-1}, & sinon \end{cases}$$

$$D_t = \begin{cases} 1, & si(|I_t - B_t| > V_t) \\ 0, & sinon \end{cases}$$

Où les indices ***t*** et ***t-1*** représentent les instants présents et précédents. ***I*** représente l'image en entrée. B représente ***sigma*** et ***V*** représente ***Delta***. ***D*** représente le résultat de la segmentation qui a pour valeur 0 ou 1.

5.5.2 Représentation de la soustraction de fond par blocs matériels

La formulation précédente doit être traduite en composants matériels spécifiques comme ceux de la section 5.3.

La figure 5-18 représente tous les blocs nécessaires pour la réalisation de cette fonction. On dénomme flux maître ou « Leader Stream » un flux dont l'horloge est maîtresse et qui contrôle donc la mise à jour des données. L'image qui arrive de la caméra est un flux maître, car c'est l'horloge de la caméra qui définit la cadence. Un flux suiveur ou « Follower Stream » est un flux récupéré depuis la mémoire à l'aide de composants tels que le « Pixel Fetcher » présenté dans la section 5.3, symbolisé ici par « Join ». Chaque bloc opérateur « Operator » est un bloc qui assure une fonctionnalité quelconque étant donné que les entrées et les sorties respectent le protocole du bus Avalon du système. C'est un bloc qui est développé sous Verilog avec la fonctionnalité spécifique nécessaire comprenant éventuellement des additions, soustraction ou multiplications. Il peut être combinatoire ou séquentiel avec un ou plusieurs niveaux de registres pour exécuter le calcul en plusieurs étapes.

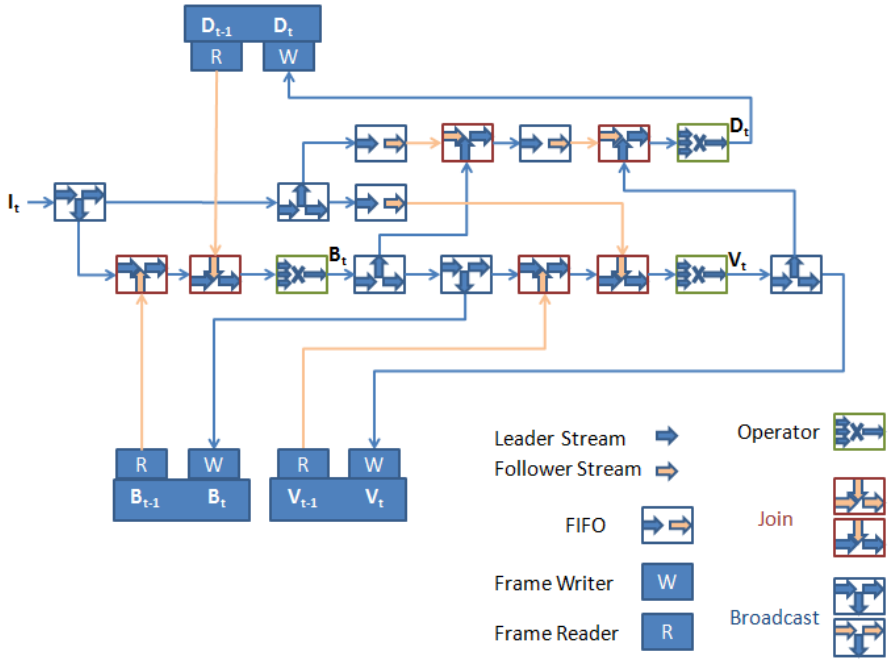

Figure 5-18 : Blocs matériels pour la soustraction de fond.

Il est évident que le nombre de blocs est assez important et fait diverger l'usage des ressources en matière de connectique, de temps de synthèse et surtout en temps de développement et vérification. Nous avons donc opté pour une optimisation spécifique qui regroupe des variables devant être écrites et lues en même temps pour les loger dans des bits différents de la même case mémoire. L'optimisation est présentée dans la figure 5-19.

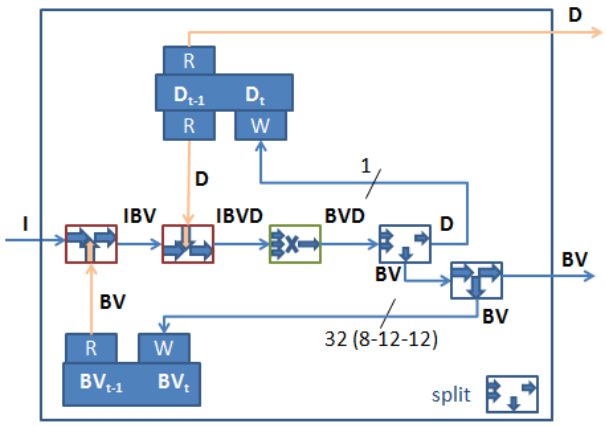

Figure 5-19 : Optimisation de la soustraction de fond.

C'est cette implémentation que l'on va réaliser à l'aide de l'outil de synthèse haut niveau QSys.

Dans la figure 5-19, on voit que les 32 bits de la mémoire (contenant BV) ont été répartis en 12 pour la variable B, 12 pour la variable V et 8 non utilisés. La variable D, ayant un unique bit, a pu se loger dans une mémoire interne à largeur 8 bits avec une adaptation entre les interfaces 8 bits et 1 bit. Cette mémoire interne a un double port d'accès qui permet de lire la variable D indépendamment des autres variables. Au final, une seule interface de mémoire externe a été utilisée au lieu de trois.

C'est le bloc « Operator »qui dans ce cas traite les données I, B, V et D pour calculer les nouvelles sorties B, V et D selon la formulation présentée dans 5.5.1.

5.5.3 Implémentation sous QuartusII - QSys

Génération automatique par QSys

On rappelle que QSys est un outil de conception haut niveau qui permet de connecter directement les signaux des bus compatibles. Avec cette version, on a pu créer un sous système de soustraction de fond, celui présenté dans la figure 5-19. Dans son implémentation QSys, il a nécessité l'usage de 17 composants de base. Il a ensuite été emboîté dans le module « BackgroudDetector » numéroté (15) dans la figure 5-20.

Connections	Name	Description	Export	Clock	Base	End	IRQ
(1)	clk_50	Clock Source					
(2)	clk_50_2	Clock Source					
(3)	altpll_0	Avalon ALTPLL		clk_50	0x08000b00	0x8000b0f	
(4)	altpll_1	Avalon ALTPLL		clk_50_2	0x08000d00	0x8000d0f	
(5)	nios2_qsys_0	Nios II Processor		clk_sys	0x08000000	0x80007ff	
(6)	cpu_mem	On-Chip Memory (RAM o...		unconnected			
(7)	sysid_qsys	System ID Peripheral		clk_sys	0x08000a00	0x8000a07	
(8)	uart	UART (RS-232 Serial Port)		clk_sys	0x08000800	0x800081f	0
(9)	sdramu1	SDRAM Controller		clk_sys	0x02000000	0x3ffffff	
(10)	CamControl_5M	Cam_Control		clk_sys	0x08000c00	0x8000c03	
(11)	Clk_ToCam	Clock Bridge		clk_cam_vga			
(12)	Clk_fromCam	Clock Bridge		exported			
(13)	CamData16	CamBayers		clk_FromCam			
(14)	WriterSync16	StreamClock					
	StreamIn_clk	Clock Input	Click to export	clk_FromCam			
	StreamIn_reset_n	Reset Input	Click to export	[StreamIn_clk]			
	StreamIn	Avalon Streaming Sink	Click to export	[StreamIn_clk]			
	StreamOut_clk	Clock Input	Click to export	clk_sys			
	StreamOut_reset_n	Reset Input	Click to export	[StreamOut_clk]			
	StreamOut	Avalon Streaming Source	Click to export	[StreamOut_clk]			
(15)	BackgroundDetector	BackgroundDetectorSD					
	clk	Clock Input	Click to export	clk_sys			
	reset	Reset Input	Click to export				
	tristate_ssram	Conduit	tristate_ssram				
	imagestreamin	Avalon Streaming Sink	Click to export	[clk]			
	backvar	Avalon Streaming Source	Click to export	[clk]			
	bkg_buffers	Avalon Memory Mapped ...	Click to export	[clk]	0x00000000	0x3fffff	
	dmask8	Avalon Streaming Source	Click to export	[clk]			
(16)	Sync_BV	StreamClock					
	StreamIn_clk	Clock Input	Click to export	clk_sys			
	StreamIn_reset_n	Reset Input	Click to export	[StreamIn_clk]			
	StreamIn	Avalon Streaming Sink	Click to export	[StreamIn_clk]			
	StreamOut_clk	Clock Input	Click to export	clk_sys_vga			
	StreamOut_reset_n	Reset Input	Click to export	[StreamOut_clk]			
	StreamOut	Avalon Streaming Source	Click to export	[StreamOut_clk]			
(17)	Sync_D	StreamClock					
	StreamIn_clk	Clock Input	Click to export	clk_sys			
	StreamIn_reset_n	Reset Input	Click to export	[StreamIn_clk]			
	StreamIn	Avalon Streaming Sink	Click to export	[StreamIn_clk]			
	StreamOut_clk	Clock Input	Click to export	clk_sys_vga			
	StreamOut_reset_n	Reset Input	Click to export	[StreamOut_clk]			
	StreamOut	Avalon Streaming Source	Click to export	[StreamOut_clk]			
(18)	BKGDisplay	BufferBKGDisplay					
	clk	Clock Input	Click to export	clk_sys_vga			
	reset	Reset Input	Click to export				
	clkvga	Clock Input	Click to export	clk_cam_vga			
	sdramu_2	Conduit	sdram_2				
	bufferstreamin	Avalon Streaming Sink	Click to export	[clk]			
	dmask_streamin	Avalon Streaming Sink	Click to export	[clk]			
	vga	Conduit	vga				

Figure 5-20 : Projet QSys du top-niveau.

Dans la figure 5-20, on perçoit la totalité des composants du système du top-niveau, dont principalement le processeur Nios2 (5) qui permet entre autres de configurer la caméra à travers un bus i²c avec le composant (10). Les données des pixels sortant de la caméra passent à travers

le composant (13). Les composants (1,2,3,4) représentent les deux horloges du système ainsi que le PLL (Phase Locked Loop) configurant leur fréquence. Le code du processeur est logé dans une mémoire interne (6) et dispose d'un composant d'identification (7) permettant de vérifier la correspondance de la version du logiciel avec celle du matériel. Le composant (8) est un port série pour le log. Le composant (9) est le contrôleur de la mémoire SDRAM. Les composants (11,12) sont des ponts permettant le passage d'un domaine d'horloge à un autre. Le module principal de soustraction de fond est le composant (15). Le composant (14) permet de lui injecter l'image. Les composants (16,17) permettent de synchroniser les données comme affiché dans la figure 5-19. Finalement, le composant d'affichage (18) est relié à un écran VGA.

Dans ce projet, il a fallu rajouter à la librairie fournie par le constructeur, des composants spécifiques. Une dizaine de modules ont été développés, contenant chacun un fichier de configuration et un fichier Verilog développé à la main (non généré). Chaque fichier contient en moyenne 200 lignes de code. La génération automatique, qui est la finalité de l'usage de cet outil, permet d'éviter de coder à la main les fichiers regroupant les composants pour les interconnecter. À titre d'exemple, pour le sous module « BackgroundDetector », un fichier Verilog de 2 800 lignes a été généré et pour le système du top-niveau, le fichier généré contient 3 500 lignes.

On voit bien comment la génération automatique nous permet de réaliser un travail dans le cadre du projet qui aurait été autrement impossible à réaliser par une personne d'autant pour le développement que pour la vérification.

Ressources matérielles et synthèse

Le tableau 5-9 récapitule les principales ressources disponibles dans le kit de développement DE2-70 utilisé pour la validation fonctionnelle de la soustraction de fond.

Composant	CycloneII - EP2C70F896C6
Éléments logiques*	68 416
Bits de mémoire RAM interne	1 152 000
Multiplicateurs	150
PLLs	4
SDRAM externe	2 x 16M x 16 bit @ 100 MHz
SSRAM externe	512K x 36 bit @ 100 MHz

Tableau 5-9 : Principales Ressources du Kit DE2-70.

(*) Un élément logique du composant Cyclone II est la plus petite entité programmable contenant principalement une LUT à 4 entrées pour les fonctions combinatoires et un registre pour les fonctions séquentielles.

Ressource	**Usage**
Éléments logiques	7 619 / 68 416 (11 %)
--Combinatoire sans registres	2 401
--Registres uniquement	2 052
--Combinatoire avec registre	3 166
LUT Fonctions à 4 entrées	2 734
LUT Fonctions à 1,2 ou 3 entrées	2 833
Total registres logiques	5 218 / 68 416 (8 %)
Total LAB**	630 / 4 276 (15 %)
Total usage mémoire interne	739 840 / 1 152 000 (64 %)
Total usage de bloc mémoire	188 / 250 (75 %)

PLLs	2 / 4 (50 %)
Horloges globales	16 / 16 (100 %)

Tableau 5-10 : Usage des ressources.

(**)Un LAB « Logic Array Block » est un ensemble de 16 éléments logiques facilitant leur interconnexion.

Si ce kit de développement a été suffisant pour l'application de soustraction de fond, l'usage des ressources a été varié selon leur nature. On résume dans le tableau 5-10 l'usage des différentes ressources.

Le tableau 5-10 nous indique également que l'usage des ressources a été déséquilibré. Uniquement 11 % des éléments logiques ont été utilisés alors que la mémoire interne a été utilisée à 64 %. Toutes les horloges globales ont été utilisées. On en conclut qu'il est facile d'utiliser un FPGA surdimensionné à l'application, mais il est compliqué de maximiser l'usage des ressources dans un projet d'intégration regroupant la totalité des fonctionnalités. Comme cette implémentation est une validation fonctionnelle de la soustraction de fond, nous n'avons pas besoin d'intégrer plus de fonctionnalités sur le même composant au sein du même projet. Par contre, ce projet permet d'évaluer les ressources nécessaires pour justifier le choix du composant final qui ciblerait l'intégration de la totalité des fonctionnalités.

5.5.4 Chronogrammes de fonctionnement

On va analyser dans cette partie deux chronogrammes de fonctionnement. Un concernant l'accès mémoire et un pour le calcul de la soustraction de fond. L'horloge de cadence est à 100 MHz.

Le premier, celui de l'accès mémoire, présenté dans la figure 5-21, est particulièrement important parce qu'il conforte la modélisation qui prévoit le partage de l'accès mémoire. Ce partage implique nécessairement un conflit d'accès à certains moments comme celui indiqué par la flèche (1). Quand la nouvelle demande d'écriture est appliquée, l'acquittement n'arrive qu'après la fin d'accès de l'écriture en cours, indiqué par la flèche (2). Pendant ce temps, il faut s'assurer que les buffers ne débordent pas.

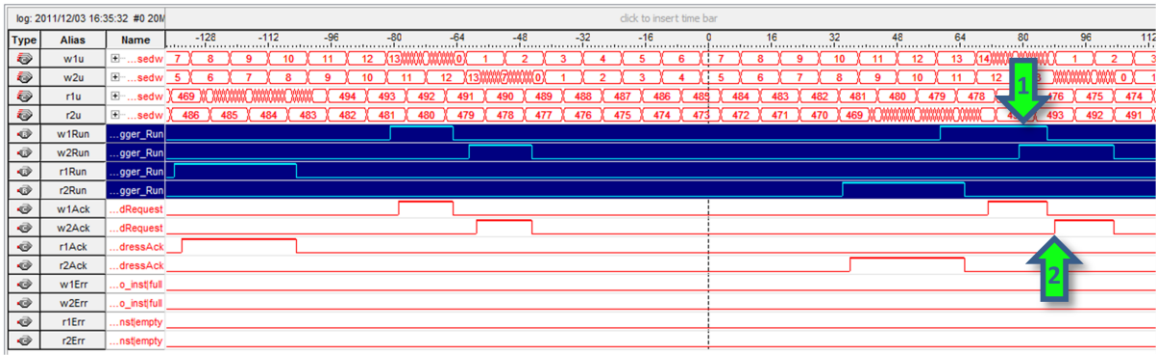

Figure 5-21 : Accès mémoire partagé.

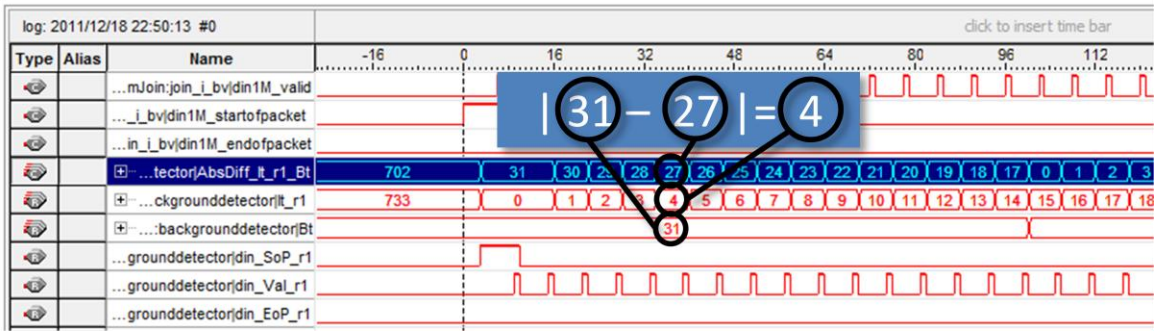

Figure 5-22 : Calcul soustraction de fond.

Dans la figure 5-22, on a un chronogramme issu du fonctionnement du module de soustraction de fond. Il concerne les variables de calculs de différence absolue. L'entrée a été alimentée par une image qui représente un compteur incrémenté d'une valeur à chaque pixel. Cela permet de vérifier le bon fonctionnement du module tel qu'indiqué par l'exemple explicité dans la figure.

5.5.5 Évaluation

Après la vérification des chronogrammes, une évaluation a été réalisée en faisant tourner le système en temps réel à des fréquences allant de 10 à 30 images par seconde en fonction de l'horloge configurée de la caméra et du temps d'exposition.

Figure 5-23 : Évaluation de la soustraction de fond.

Cette évaluation préliminaire sert uniquement à s'assurer du bon fonctionnement de l'algorithme et de la fonctionnalité intégrée. Son usage pour la détection de posture humaine relève du cadre d'intégration de toutes les autres fonctionnalités dans un système complet.

5.6 Conclusion

Dans ce chapitre, on a commencé par analyser les différentes architectures pour montrer l'intérêt et la nécessité de l'utilisation de chacune d'entre elles. Il n'y en a pas une meilleure qu'une autre, mais chacune est appropriée à un contexte applicatif bien défini.

On a transposé cette analyse sur l'application de détection de posture qui est l'objet de ce manuscrit, et cela nous a amené à trois solutions complémentaires qui sont le PC, le GPU et le circuit FPGA.

La partie développée sur PC nous a permis une grande flexibilité assurant une étude complète des paramètres algorithmiques. Elle constitue une plate-forme de développement incontournable dans les phases de développement du projet. Elle a été jointe à la plate-forme d'acquisition qui a permis de fournir les données d'apprentissage.

La partie développée sur GPU a permis l'accélération la plus importante avec un temps d'apprentissage minimum, mais on s'est contenté d'une unique fonctionnalité à titre comparatif. Les GPU nécessitent un environnement PC gourmand en ressources énergétiques et cela a découragé la continuité de l'investigation de cette approche.

La partie développée sur FPGA a retenu notre attention vu qu'elle permet à un système de fonctionner avec des ressources énergétiques limitées tout en assurant une fonctionnalité complexe. Nous nous sommes donc investis dans une première phase pour le développement de l'architecture qui a donné lieu à une publication (Filali et al. 2011) et qui a servi dans d'autres projets. Dans la deuxième phase, nous avons assuré toutes les étapes de développement de la soustraction de fond. Nos perspectives sont bien évidemment de procéder à l'intégration de la totalité des fonctions de détection de posture dans un même système.

Chapitre 6 : Conclusion et perspectives

Ce manuscrit présente le contexte et les motivations associés à ces travaux démarrés en Octobre 2009. Certes le Kinect a été disponible fin 2010 mais les travaux associés sur la reconstruction de posture ont été publiés en 2011 et 2012. Nous avons donc dû adapter notre feuille de route pour prendre en compte ces avancées technologiques et méthodologiques majeures. Nos travaux ont alors focalisé sur l'implémentation d'une nouvelle approche algorithmique de fusion bas niveau qui utilise plusieurs capteurs de profondeur pour en tirer une meilleure précision et robustesse. Puis, nous avons visé une partie d'intégration matérielle, car elle constitue l'étape la plus importante dans la mise en place de produits.

Le chapitre 2 décrit notre plate-forme multi-capteurs et leur couplage avec un système de capteur de mouvement commercial par marqueurs passifs (MOCAP). Le processus lourd d'étalonnage du système complet (spatial, temporel), les jeux de données acquises avec leur vérité terrain MOCAP associés sont ainsi décrits. Ces données avec vérité-terrain représentent à nos yeux un intérêt non négligeable pour la communauté Vision par Ordinateur car il n'existait pas en 2012 de « benchmarks » avec vérité terrain permettant de comparer les approches de reconstruction de postures humaines.

Le chapitre 3 décrit notre approche par voxellisation issue des diverses vues ; il inclut à ce titre les deux contributions majeures de la thèse. Notre approche se démarque des travaux de Shotton et al. qui sont mono-capteurs et s'appuient sur des descripteurs dans l'image de disparité. Nous proposons une étude pointue sur le réglage (« tuning ») des paramètres libres sous-jacents à la formalisation de notre approche et au-delà propre aux « random forest ». La littérature est pas ou très peu loquace sur ce sujet alors que ce « tuning », comme démontré, influence grandement les performances globales de la méthode.

Le chapitre 4 illustre et discute les performances qualitatives et quantitatives de notre approche et se compare avec l'approche OpenNI inspirée de Shotton et al. Plusieurs critères, en termes de taux de classification des voxels et précision de reconstruction des parties, sont évalués. Certes, notre approche multi-capteurs est logiquement supérieure mais nous avons ici quantifié les gains apportés et démontré l'attrait d'une fusion bas niveau multi-capteurs.

Le chapitre 5 focalise sur l'état de l'art de l'intégration matérielle et plus spécifiquement sur l'intégration sur circuit FPGA. Nous avons présenté les architectures et les méthodologies associées, ce qui nous a permis de défendre notre approche. L'approche de conception du type « rencontre au point milieu » a consisté d'abord à modéliser le système depuis le haut niveau et définir le fonctionnement et les étapes du processus. Ensuite, considérer le bas niveau pour spécifier l'environnement de développement et proposer l'approche fondée sur des modules

unitaires permettant un développement hiérarchique. Le placement des ressources, enfin, concrétise la rencontre des deux approches. On a comparé le placement d'une fonction de soustraction de fond sur des ressources différentes CPU, GPU et FPGA. Et on a fini par l'étude complète d'implémentation sur FPGA de cette fonctionnalité.

Nos principales **contributions** sont :

(i) L'acquisition d'une base de données avec vérité-terrain. La complexité de cette étape, la rareté des systèmes de MOCAP et l'arrivée récente des capteurs de profondeur sont les éléments qui expliquent l'inexistence d'une telle base de données dans la littérature.

(ii) La classification de données voxellisées par des forêts de décision. Ces données ont été obtenues grâce à la fusion en bas niveau de plusieurs capteurs de profondeur. Cette étape a été critique par la nature même des données qui sont massives.

(iii) La caractérisation des paramètres libres du « random forest ». Peu de travaux explicitent clairement le « tuning » de ces paramètres alors que ceux-ci influencent les performances. Il est également possible de jouer sur les compromis entre qualité et temps de traitement.

(iv) Un environnement de développement avec les fonctionnalités nécessaires pour créer des applications de traitement d'images. Cette méthodologie est arrivée juste après la mise à disposition par les constructeurs d'outils de synthèse de haut niveau (SOPC et QSys). Nous les avons complétés par des composants de base permettant la concentration sur un niveau d'ordre supérieur. Cette contribution a été déclinée par d'autres doctorants sur plusieurs projets. Nous l'avons-nous mêmes mise à profit pour l'intégration de la fonctionnalité de soustraction de fond.

Pour ces travaux, les **perspectives sont multiples**.

Pour limiter les confusions entre **gauche et droite** (phénomène de « flip») il serait opportun de filtrer temporellement les reconstructions obtenues image par image pour éviter les sauts improbables entre deux images. Dans cette optique, il serait intéressant de comparer avec une stratégie de **fusion haut niveau** qui considère le squelette inféré pour chaque capteur et appliquer un filtrage au niveau haut (thèse en cours de Jean Thomas Masse au LAAS). Cette démarche permet de profiter plus facilement des nouvelles versions de capteurs mis à disposition sur le marché sans avoir à repasser par des étapes d'apprentissage.

Les bases de données peuvent être largement enrichies par des **images de synthèse,** plus faciles à obtenir. Même si nous avons utilisé des modèles virtuels de personnes très fidèles à la morphologie humaine[52], il manquait la variété et la qualité du rendu du style vestimentaire, ainsi qu'une chaîne de rendu de qualité qui considère des modèles de bruits proches des capteurs optiques. Passé outre ces réserves, il serait donc intéressant d'exploiter des images de synthèse.

[52] http://www.makehuman.org/

Un autre enjeu porterait sur la **parallélisation d'algorithmes d'apprentissage**. Si dans l'état de l'art une grappe de centaines de PC a été utilisée, c'est que cela donne un avantage pour pouvoir apprendre des bases de données qui sont de l'ordre des centaines de milliers de postures et non pas juste de quelques milliers comme c'est le cas actuellement. Cette parallélisation peut se faire sur des grappes de PC, ou plutôt tirer profit de la nouvelle technologie de GPUs qui ont déjà atteint à ce jour 2 880 cores[53].

Au niveau de l'intégration matérielle, la perspective est naturellement d'intégrer la totalité des fonctionnalités de détection de posture sur le même composant, et de maintenir une librairie de classification par arbre de décision sur FPGA. L'accélération de l'apprentissage sur FPGA est une autre problématique pouvant ouvrir de nouvelles perspectives pour l'apprentissage.

Par ailleurs citons que ces travaux se poursuivent via des projets impliquant le LAAS-CNRS et des industriels afin développer un prototype commercialisable. Dans cette perspective, les facteurs de coût, de flexibilité d'installation, de prise en main sont à considérer.

Enfin pour terminer, il serait opportun d'utiliser notre solution pour alimenter un module de reconnaissance d'activités humaines, d'actions coopératives de plusieurs personnes ou d'interaction homme-machine.

[53] http://www.nvidia.com/object/tesla-workstations.html

Table des figures

Figure 1-1 : L'édition 2005 de l'ITRS (International Technology Roadmap for Semiconductors) du nombre de processeurs contenus dans un système embarqué. 14
Figure 1-2 : Capteurs de profondeur grand public. 16
Figure 1-3 : Principe de fonctionnement du Kinect. 17
Figure 2-1 : caractéristiques de l'image de profondeur (a) grande différence entre les offsets (b) petite différence (Shotton et al. 2011a). 27
Figure 2-2 : Image de profondeur – Image classifiée en parties du corps – Joints vue de face – Joints vue de côté – joint vue de dessus (Shotton et al. 2011a). 27
Figure 2-3 : agrégation des votes des pixels lors du test (Girshicky et al. 2011). 28
Figure 2-4 : The vitruvian manifold - (Taylor et al. 2012). 28
Figure 2-5 : Différents modèles de régression - (Sun et al. 2012). 29
Figure 2-6 : 1 – Échiquier d'étalonnage Vision. 2 – Caméra de profondeur (xtion). 3 - Caméras IR du système MOCAP. 4 – L-Frame d'étalonnage du système MOCAP. 32
Figure 2-7 : Zoom sur une caméra infrarouge du système MOCAP. 33
Figure 2-8 : Vue d'ensemble du correspondant virtuel de l'environnement de capture (1 segment = 20cm). 33
Figure 2-9 : Placement des marqueurs. 34
Figure 2-10 : Squelette du système MOCAP. 35
Figure 2-11 : Équerre d'étalonnage. 36
Figure 2-12 : Précision de re-projection (en pixels) des marqueurs sur les plans RGB des capteurs de profondeur. 37
Figure 2-13 : Synchronisation du MOCAP avec un mouvement rapide. 38
Figure 2-14 : Nom des marqueurs de la capture MOCAP. 38
Figure 3-1 : Synopsis de notre approche. 43
Figure 3-2 : Notre approche vs. Shotton et al. - (Filali et al. 2013). 44
Figure 3-3 : Segmentation basée uniquement sur la variation de l'image de la profondeur. 44
Figure 3-4 : Voxellisation d'une silhouette associée à une posture humaine. 45
Figure 3-5 : Exemples de labellisation des parties corporelles. 45
Figure 3-6 : Échantillonnage de l'espace et voxellisation. 46
Figure 3-7 : Voxellisation par vision passive vs. vision active. 47
Figure 3-8 : Descripteur 3D de taille fixe. 48
Figure 3-9 : Architecture du réseau de neurones (Chaabani et al. 2012). 49
Figure 3-10 : Structure et terminologie d'un arbre de décision. 52
Figure 3-11 : Affichage d'un arbre de décision. 53
Figure 3-12 : Principe de l'algorithme Meanshift. 57
Figure 3-13 : Nuage de 10K voxels tirés aléatoirement. 58
Figure 3-14 : Labellisation des parties du corps de référence (a) = volume non labellisé. (b) = parties du corps principales. (c) = parties principales et intermédiaires. 58
Figure 3-15 : mAP en fonction du nombre de candidats d'. 61
Figure 3-16 : mAP et Classification vs. sous-échantillonnage. 62
Figure 3-17 : Classification et mAP vs. profondeur maximale. 63
Figure 3-18 : Temps d'apprentissage et temps de classification vs. profondeur maximale. 63

Figure 3-19 : Classification et mAP vs. seuil du gain d'information. 64
Figure 3-20 : Nombre d'échantillons vs. profondeur pour différentes valeurs du seuil de gain d'information. 65
Figure 3-21 : Nombre total de nœuds vs. profondeur pour différentes valeurs du seuil de gain d'information. 65
Figure 3-22 : Taille de la fenêtre des Vecteurs de descripteurs – UniNorme. 66
Figure 3-23 : Nombre d'arbres dans la forêt – UniNorm. 67
Figure 3-24 : Exemples de classification selon le nombre d'arbres. 68
Figure 3-25 : Comparaison des techniques de Vote selon le mAP. 69
Figure 3-26 : Comparaison des techniques de vote selon la classification. 69
Figure 4-1 : Nombre de voxels par partie corporelle pour la posture M2-99. 73
Figure 4-2 : Volume de travail pour la posture M2-99. 73
Figure 4-3 : Nombre d'échantillons des parties corporelles à apprendre et temps d'apprentissage en fonction des étapes d'apprentissage. 75
Figure 4-4 : Vérité terrain avec labellisation des parties corporelles et centres des joints – M3-236. 77
Figure 4-5 : Différenciation entre gauche et droite : ambiguïté. 77
Figure 4-6 : Posture complexe, avec erreur de reconstruction. 78
Figure 4-7 : Pourcentage des postures vs. ratio de classification des voxels. 79
Figure 4-8 : Précision de l'estimation pour la posture M3-236. 82
Figure 4-9 : Évaluation qualitative d'OpenNI. 83
Figure 4-10 : IRSS35-C3 :BPR vs. OpenNI. 84
Figure 4-11 : Posture simple – Posture en T (IRSS35-C3-121). 85
Figure 4-12 : Erreur de « flip » gauche-droite (IRSS35-C3-147). 85
Figure 4-13 : Fusion de la main (IRSS35-C3-473). 86
Figure 4-14 : Fusion des mains BPR (IRSS35-C3-917). 86
Figure 4-15 : Posture complexe (IRSS35-C3-1475). 87
Figure 4-16 : Comparaison quantitative : notre approche vs. OpenNI (séquence 1). 88
Figure 4-17 : Comparaison quantitative : BPR vs. OpenNI (séquence 2). 89
Figure 5-1 : Répartition des architectures des calculateurs. 94
Figure 5-2 : Utilisation du kinect. 98
Figure 5-3 : PS1080 (Primesense). 100
Figure 5-4 : Diagramme de définition des blocs. 100
Figure 5-5 : Diagramme d'activité de l'unité de traitement temps réel. 101
Figure 5-6 : a - Banc Stéréo 100 Hz b – Robot équipé du système de vision (LAAS-CNRS). 103
Figure 5-7 : Plate-forme de l'application multi caméras – a – caméras – b – adaptateurs – c – Kit de développement Stratix III (TerasicD5M, AlteraDevKit). 104
Figure 5-8 : Caméra vers Avalon. 105
Figure 5-9 : Transformation des flux. 106
Figure 5-10 : Récupération des pixels. 106
Figure 5-11 : principe de fonctionnement de la correction de distorsion. 108
Figure 5-12 : Transformation homographique. 108
Figure 5-13 : Accès multi caméras. 110
Figure 5-14 : Concaténation de modules de traitement – (a) connexion des modules – (b) Image source distordue – (c) Image rectifiée et transformée. 111

Figure 5-15 : Architecture PCArchitecture PC multi Capteurs (PC HP-XtionProLive). 112
Figure 5-16 : Architecture d'un GPU : (gauche) Exécution – (droite) Mémoires (Nvidia). 114
Figure 5-17 : Architecture embarquée sur FPGA (AlteraDevKit). 115
Figure 5-18 : Blocs matériels pour la soustraction de fond. 119
Figure 5-19 : Optimisation de la soustraction de fond. 119
Figure 5-20 : Projet QSys du top-niveau. 120
Figure 5-21 : Accès mémoire partagé. 122
Figure 5-22 : Calcul soustraction de fond. 123
Figure 5-23 : Évaluation de la soustraction de fond. 123

Liste des tableaux

Tableau 1-1 : Exemple de librairies associées aux capteurs de profondeur et leurs fonctions (Cruz et al. 2012)..... 18
Tableau 1-2 : Comparaison du Kinect avec Hofmann et al 2009. 18
Tableau 2-1 : Récapitulation des caractéristiques des bases de données disponibles. 31
Tableau 2-2 : Caractéristiques des caméras du système de MOCAP. 33
Tableau 2-3 : Détails sur les bases de données acquises. 39
Tableau 3-1 : Temps de classification avec k-PPV (Base Synth-L300P10). 50
Tableau 3-2 : Classification et précision en fonction de la taille du descripteur (Base de données RHE-L7P1) 50
Tableau 3-3 : Récapitulation des paramètres/variables des « Random Forests ». 56
Tableau 3-4 : Paramètres par défaut utilisés. 59
Tableau 3-5 : Résultats et paramètres d'arbre d'apprentissage. 60
Tableau 3-6 : Paramètres de l'arbre en fonction du sous-échantillonnage. 62
Tableau 3-7 : Temps d'apprentissage et de classification vs. nombre d'arbres. 67
Tableau 3-8 : Réglage (« tuning ») des divers paramètres libres. 70
Tableau 4-1 : Base de données utilisée. 72
Tableau 4-2 : Valeurs des paramètres libres. 74
Tableau 4-3 : Caractérisation des « random forest » appris. 74
Tableau 4-4 : Matrice de confusion des parties corporelles principales pour la posture M3-236.79
Tableau 4-5 : Matrice de confusion pour la posture M3-236. 80
Tableau 4-6 : Statistiques sur l'estimation des centres des parties corporelles. 81
Tableau 4-7 : Pourcentage des centres prédits (mAP) avec une erreur inférieure au seuil. 81
Tableau 4-8 : Étude comparative : ases de données utilisées..... 82
Tableau 4-9 : Étude comparative : valeurs des paramètres libres. 83
Tableau 4-10 : Seconde base d'étude supplémentaire. 88
Tableau 5-1 : Ressources de différentes plates-formes. 97
Tableau 5-2 : Comparaison des transformations. 107
Tableau 5-3 : Paramètre des flux et bande passante associée. 109
Tableau 5-4 : Placement des fonctions. 112
Tableau 5-5 : Caractéristiques de la carte graphique utilisée. 114
Tableau 5-6 : Résultats de comparaison OpenCL sur CPU-GPU-FPGA (Altera- wp-01173-opencl). 116
Tableau 5-7 : Temps d'exécution sur multiples plates-formes. 117
Tableau 5-8 : Détails des opérations GPU. 117
Tableau 5-9 : Principales Ressources du Kit DE2-70. 121
Tableau 5-10 : Usage des ressources. 122

Glossaire

ASIC	Application Specific Integrated circuit
BSD	Berkeley Software Distribution (contexte : licence BSD)
CISC	Complex Instruction Set Computing
DUT	Device Under Test
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
RISC	Reduced Instruction Set Computing
LAB	Logic Array Block (Altera specific, group of 16 LE)
LE	Logic Element (Altera specific unitary FPGA element)
mAP	Mean Average Precision
MOCAP	Nom donné au matériel et au logiciel donné par le fournisseur à l'équipement de capture de mouvement
NiTE	Natural interaction Engine. C'est un "Middleware" (Inergiciel) pour OpenNI.
OpenNI	Open Natural Interaction. C'est le nom de l'organisation.
PLL	Phase-Locked Loop
SOPC	System On a Programmable Chip
LUT	Look Up Table

Bibliographie

Abd-El-Barr, M. & El-Rewini, H., 2004. *Fundamentals of computer organization and architecture* Vol. 31., Wiley-Interscience.

Agarwal, A. & Triggs, B., 2006. Recovering 3D human pose from monocular images. *IEEE transactions on pattern analysis and machine intelligence*, 28(1), pp.44–58.

Belbachir, A.N., 2010. *Smart Cameras*, Springer.

Berger, K. et al., 2011. Markerless Motion Capture using multiple Color-Depth Sensors. In *Vision, Modeling and Visualisation*. p. 3.

Bodenheimer, B. et al., 1997. The Process of Motion Capture : Dealing with the Data. In *Computer Animation and Simulation*. Budapest, Hongrie, pp. 3–18.

Boulay, B., 2007. *Human Posture Recognition for Behaviour Understanding*. Université de Nice-Sophia Antipolis.

Bradski, G. & Kaehler, A., 2008. *Learning OpenCV*, O'Reilly.

Bradski, G.R. & Pisarevsky, V., 2000. Intel ' s Computer Vision Library : Applications in calibration , stereo , segmentation , tracking , gesture , face and object recognition . In *IEEE Conference on Computer Vision and Pattern Recognition*. pp. 796–797 vol.2.

Breiman, L., 2001. Random forests. *Machine Learning*, 45(1), pp.5–32.

Brown, D.C., 1971. Close-range camera calibration. *Photogrammetric engineering*, 37(8), pp.855–866.

Brubaker, M. a., Fleet, D.J. & Hertzmann, A., 2009. Physics-Based Person Tracking Using the Anthropomorphic Walker. *International Journal of Computer Vision*, 87(1-2), pp.140–155.

Budiu, M. et al., 2011. Parallelizing the Training of the Kinect Body Parts Labeling Algorithm. In *Big Learning: Algorithms, Systems and Tools for Learning at Scale*. pp. 1–6.

Chaabani, H. et al., 2012. Body Pixel Classification by Neural Network. In *International Conference on Intelligent Robotics and Applications*. pp. 494–502.

Chase, J. et al., 2008. Real-Time Optical Flow Calculations on FPGA and GPU Architectures: A Comparison Study. In *International Symposium on Field-Programmable Custom Computing Machines*. Ieee, pp. 173–182.

Cheng, S.Y. & Trivedi, M.M., 2007. Articulated Human Body Pose Inference from Voxel Data Using a Kinematically Constrained Gaussian Mixture Model. In *CVPR Workshop on Evaluation of Articulated Human Motion and Pose Estimation*.

Cheng, Y., 1995. Mean Shift , Mode Seeking , and Clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8), pp.790–799.

Cheung, G.K.M., Baker, S. & Kanade, T., 2003. Shape-From-Silhouette of Articulated Objects and its Use for Human Body Kinematics Estimation and Motion Capture. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Chrlstodoulou, C. et al., 2002. Wireless Telemedicine Systems : An Overview. *Antennas and Propagation Magazine, IEEE*, 44(2), pp.143–153.

Cohen-Or, D., and Kaufman, A., 1995. Fundamentals of Surface Voxelization. *Graphical Models and Image Processing*.

Corazza, S. et al., 2010. Markerless Motion Capture through Visual Hull, Articulated ICP and Subject Specific Model Generation. *International Journal of Computer Vision*, 87(1-2), pp.156–169.

Criminisi, A., 2011. Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. *Foundations and Trends® in Computer Graphics and Vision*, 7(2-3), pp.81–227.

Cruz, L., Lucio, D. & Velho, L., 2012. Kinect and RGBD Images: Challenges and Applications. In *IEEE Conference on Graphics, Patterns and Images Tutorials*. Ieee, pp. 36–49.

Dalal, N. & Triggs, B., 2005. Histograms of Oriented Gradients for Human Detection. *IEEE Conference on Computer Vision and Pattern Recognition*, 1.

Deutscher, J. & Reid, I., 2005. Articulated Body Motion Capture by Stochastic Search. *International Journal of Computer Vision*, 61(2), pp.185–205.

Ejiri, M., 2001. Robotics and machine vision for the future-an industrial view. In *Advanced Intelligent Mechatronics*. pp. 917–922.

Esmaeilzadeh, H. et al., 2011. Dark Silicon and the End of Multicore Scaling. In *International Symposium on Computer Architecture*. pp. 365–376.

Fathi, A. & Greg, M., 2007. Human Pose Estimation using Motion Exemplars. In *IEEE International Conference on Computer Vision*. pp. 1–8.

Filali, W. et al., 2013. Human motion capture using 3D reconstruction based on multiple depth data. In *IEEE International Conference on Systems, Man, and Cybernetics*.

Filali, W. et al., 2011. SOPC components for real time Image processing : Rectification , Distortion correction and Homography. In *IEEE Workshop on Electronics, Control, Measurement and Signals*.

Fofi, D., Sliwa, T. & Voisin, Y., 2004. A Comparative survey on invisible structured light. In *SPIE Electronic Imaging - Machine Vision Applications in Industrial Inspection*. pp. 90–97.

FONTMARTY, M., 2008. *Vision et filtrage particulaire pour le suivi tridimensionnel de mouvement humain. Applications à la Robotique*. Université de Toulouse.

Frank Ghenassia, 2005. *Transaction-Level Modeling with SystemC - TLM Concepts and Applications for Embedded Systems*, Springer.

Friedenthal, S., Moore, A. & Steiner, R., 2009. *OMG Systems Modeling Language Tutorial*,

Gac, N. et al., 2008. High Speed 3D Tomography on CPU, GPU, and FPGA. *EURASIP Journal on Embedded Systems*, 2008(1), p.930250.

Gall, J. et al., 2009. Motion Capture Using Joint Skeleton Tracking and Surface Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*. pp. 1746–1753.

Ganapathi, V. et al., 2010. Real time motion capture using a single time-of-flight camera. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp.755–762.

Girshicky, R. et al., 2011. Efficient Regression of General-Activity Human Poses from Depth Images. In *IEEE International Conference on Computer Vision*. IEEE, pp. 415–422.

Gond, L. et al., 2008. A 3D shape descriptor for human pose recovery. In *International Workshop on Articulated Motion and Deformable Objects*. pp. 370–379.

Hasler, N. et al., 2009. Markerless Motion Capture with Unsynchronized Moving Cameras. In *IEEE Conference on Computer Vision and Pattern Recognition*. pp. 224–231.

Heath, S., 2002. *Embedded Systems Design*, Newnes.

Hennessy, J.L. & Patterson, D.A., 2012. *Computer architecture: a quantitative approach*, Elsevier.

Hofmann, M. & Gavrila, D.M., 2009. Multi-view 3D human pose estimation combining single-frame recovery, temporal integration and model adaptation. *IEEE Conference on Computer Vision and Pattern Recognition*, pp.2214–2221.

Howard, I.P., 2012. *Perceiving in Depth, Volume 1: Basic Mechanisms*, Press, Oxford university.

Kalarot, R. & Morris, J., 2010. Comparison of FPGA and GPU implementations of real-time stereo vision. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, pp.9–15.

Kukkala, P. et al., 2005. UML 2 . 0 Profile for Embedded System Design. In *Design, Automation and Test in Europe*. pp. 710–715.

Laganière, R., 2011. *OpenCV 2 Computer Vision Application Programming Cookbook*, Packt Pub Limited.

Lee, S., Choi, O. & Horaud, R., 2013. *Time-of-Flight Cameras : Principles , Methods and Applications*, Springer.

Leibson, S., 2006. The Future of Nanometer SOC Design. In *International Symposium on System-on-Chip*. pp. 1–6.

Li, K., Dai, Q. & Xu, W., 2011. Markerless Shape and Motion Capture from Multiview Video Sequences. In *IEEE Transactions on Circuits and systems for Video Technology*. pp. 320–334.

Liang-qun, L., Hong-bing, J. & Jun-hui, L., 2005. The iterated extended kalman particle filter. *IEEE International Symposium on Communications and Information Technology, 2005. ISCIT 2005.*, 2, pp.1172–1175.

Lipton, A.J. et al., 2000. *A System for Video Surveillance and Monitoring*, Carnegie Mellon University, the Robotics Institute.

Lyon, D., 2007. *Surveillance Studies: An Overview*, Polity Press.

Manzanera, A. & Richefeu, J.., 2004. A robust and computationally efficient motion detection algorithm based on sigma–delta background estimation. In *Proceedings of the Fourth Indian Conference on Computer Vision, Graphics & Image Processing*. Kolkata, India, pp. 46–51.

Martin, G., 2002. UML for Embedded Systems Specification and Design : Motivation and Overview. In *Conference on Design, automation and test in Europe*. p. 773.

Martin, G., Lavagno, L. & Louis-guerin, J., 2001. Embedded UML : a merger of real-time UML and co-design. In *International Symposium on Hardware/software codesign*. pp. 23–28.

Moeslund, T.B., Hilton, A. & Krüger, V., 2006. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2-3), pp.90–126. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1077314206001263 [Accessed March 4, 2012].

Moore, G.E., 1965. Cramming more components onto integrated circuits. *IEEE Solid-State Cicuits Newsletter*, 38(8), p.114.

Muja, M. & Lowe, D.G., 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*. pp. 331–340.

Piccardi, M., 2004. Background subtraction techniques: a review. In *IEEE International Conference on Systems, Man, and Cybernetics*.

Plagemann, C. & Koller, D., 2010. Real-time Identification and Localization of Body Parts from Depth Images. In *IEEE International Conference on Robotics and Automation*.

Poppe, R., 2007. Vision-based human motion analysis: An overview. *Computer Vision and Image Understanding*, 108(1-2), pp.4–18.

Riccobene, E. et al., 2005. A SoC Design Methodology Involving a UML 2 . 0 Profile for SystemC. In *Design, Automation and Test in Europe*. pp. 704–709 Vol. 2.

Rogez, G. et al., 2008. Randomized trees for human pose detection. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 1–8.

Sachpazidis, D.I., 2008. *Image and Medical Data Communication Protocols for Telemedicine and Teleradiology*.

Shotton, J. et al., 2011a. Real-time human pose recognition in parts from single depth images. In *IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, pp. 1297–1304.

Shotton, J. et al., 2011b. Real-Time Human Pose Recognition in Parts from Single Depth Images: Supplementary Material. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Sigal, L. et al., 2004. Tracking Loose-limbed People. *IEEE Conference on Computer Vision and Pattern Recognition*, (Figure 1).

Sigal, L., Balan, A.O. & Black, M.J., 2010. HumanEva: Synchronized Video and Motion Capture Dataset for Evaluation of Articulated Human Motion. *International Journal of Computer Vision*, 87(March), pp.4–27.

Sun, M., Kohli, P. & Shotton, J., 2012. Conditional regression forests for human pose estimation. *IEEE Conference on Computer Vision and Pattern Recognition*, pp.3394–3401.

Sundaresan, A. & Chellappa, R., 2005. Markerless Motion Capture using Multiple Cameras. In *Computer Vision for Interactive and Intelligent Environment*. pp. 15–26.

Taylor, J. et al., 2012. The Vitruvian Manifold : Inferring Dense Correspondences for One-Shot Human Pose Estimation Optimization of Model Parameters. In *IEEE Conference on Computer Vision and Pattern Recognition*. pp. 103–110.

Tenorth, M., Bandouch, J. & Beetz, M., 2009. The TUM Kitchen Data Set of everyday manipulation activities for motion tracking and action recognition. In *IEEE International Conference on Computer Vision Workshops*. Ieee, pp. 1089–1096.

Urtasun, R. & Fua, P., 2004. 3D Human Body Tracking Using Deterministic Temporal Motion Models. In *European Conference on Computer Vision*. pp. 92–106.

Zhang, L., Cremers, D. & Lee, D., 2012. Real-time Human Motion Tracking using Multiple Depth Cameras. In *IEEE International Conference on Intelligent Robots and Systems*. pp. 2389–2395.

Zhuang, H. et al., 2012. 3D depth camera based human posture detection and recognition Using PCNN circuits and learning-based hierarchical classifier. *The International Joint Conference on Neural Networks*, pp.1–5.