Vision multi-caméra pour la détection d'obstacles sur un robot de service: des algorithmes à un système intégré

Mario Ibarra Manzano

THESE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *INSA de Toulouse*

Discipline ou spécialité : *Conception des circuits micro-électroniques, microsystèmes, nanosystèmes*

Présentée et soutenue par *Mario Alberto IBARRA MANZANO*

Le *7 Janvier 2011*

Titre : *Vision multi-caméras pour la détection d'obstacles sur un robot de service : des algorithmes à un système intégré.*

JURY

Mohamed AKIL, Professeur, Groupe ESIEE Paris, Noisy Le Grand
Jean-Louis BOIZARD, Maître de Conférence, LAAS-CNRS, Université de Toulouse II Le Mirail
Philippe COUSSY, Maître de Conférence, Université de Bretagne Sud, Lorient
Michel DEVY, Directeur de Recherche CNRS, LAAS-CNRS, Toulouse
Jean-Yves FOURNIOLS, Professeur, LAAS-CNRS, INSA de Toulouse
Dominique GINHAC, Professeur, Université de Bourgogne, Dijon

Ecole doctorale : *GEET*
Unité de recherche : *LAAS - CNRS*
Directeur(s) de Thèse : *Jean-Yves FOURNIOLS, Jean-Louis Boizard*
Rapporteurs : *Mohamed Akil, Dominique Ginhac*

Remerciements

Tout d'abord je tiens à remercier mes directeurs de thèse, Jean-Yves Fourniols et Jean-Louis Boizard, qui m'ont soutenus constamment durant la préparation de ce doctorat. Un grand merci à Jean-Louis pour les nombreuses corrections faites sur le manuscrit, pour les discussions fréquentes sur les travaux en cours, sur les blocages, sur les outils disponibles…

Ensuite je tiens à remercier Mr Michel Devy, qui m'a accueilli lors de mon arrivée au LAAS-CNRS, qui est à l'origine de la préparation de cette thèse du fait des coopérations qu'il maintient avec mon université d'origine, dans l'Etat de Guanajuato au Mexique. Malgré un agenda plus que chargé, il a toujours été de bon conseil, et est à l'origine du dispositif appelé « ceinture de micro-caméras » sur lequel j'ai travaillé durant ces trois dernières années.
.
Je tiens à remercier Mohamed Akil et Dominique Ginhac qui ont accepté d'être rapporteurs et dont les remarques positives et constructives m'ont encouragé à terminer ce travail plus sereinement.

Je remercie également Philippe Coussy, créateur du logiciel de développement sur FPGA appelé GAUT, pour l'intérêt qu'il a porté à ce travail en acceptant d'en être un examinateur. Par ailleurs il m'a accueilli à Lorient dans l'Université de Bretagne Sud, pendant une semaine lors de ma première année de thèse. Ce séjour a été pour moi, une très bonne introduction à l'utilisation d'outils de synthèse de haut niveau afin de développer des applications sur FPGA.

Mes remerciements vont également vers l'un des hommes de l'ombre de cette thèse, Mr Pierre Lacroix, assistant-ingénieur dans le service II du LAAS, qui m'a emmené un support technique important pour que je dispose d'un démonstrateur afin de valider mes travaux.

Je ne saurai oublier mes collègues doctorants, particulièrement Diego et Wassim qui travaillent aussi sur la thématique Adéquation Algorithmes-Architecture ; le travail commun avec eux m'a permis d'enrichir les capacités fonctionnelles du système de détection que j'ai développé.

Enfin, je ne peux terminer ces remerciements sans remercier celle que je ne remercierais jamais assez, celle avec qui j'ai partagé ce séjour de trois ans qui s'achève avec la soutenance de nos thèses respectives de manière synchronisée, ma compagne Dora Luz !!

Je dédie cette thèse à mon père
J. Porfirio Ibarra-Espinosa †,
qui fut le premier de tous mes professeurs,
de qui j'ai appris les meilleures leçons
et avec qui j'ai eu les meilleures discussions.

Table des matières

Table des figures

Première partie

Introduction et Algorithmique

Chapitre 1

Introduction

Sommaire

1.1 Motivations des travaux

1.1.1 Contexte robotique

Les études présentées dans ce manuscrit de thèse sont consacrées à la problématique de la navigation visuelle d'un robot de façon autonome et robuste, dans des environnements d'intérieur structuré, dynamique et fortement encombré du fait de la présence humaine. Plus précisément, nous nous sommes intéressés à l'étude du déplacement d'un robot de service dans des milieux urbains publics, comme les centres commerciaux, les stations de transport, les musées ou autres, en percevant visuellement les principaux objets qui l'entourent. Pour un robot de nettoyage par exemple, l'environnement pourrait ressembler à celui synthétisé dans la figure 1.1 en utilisant le simulateur MORSE [73].

La perception epuis es capteurs embarqués sur un robot consiste à analyser des dones sensorielles acquises sur l'entourage du robot, pour décrire sa position, l'état et les caractéristiques du lieu et des objets qui pourraient perturber l'exécution de mouvements par le robot. Cette analyse peut être améliorée si l'environnement lui-même est instrumenté (intelligence ambiante, Indoor GPS, identification d'objets par des RFID,etc.), ou en exploitant des nouveaux capteurs plus rapides ou plus précis (caméras 3D à temps de vol, systèmes de perception dédié, parmi d'autres).

Toutefois, c'est le type de scénario et l'application qui va conduire à choisir tel ou tel capteur à utiliser sur le robot. Dans notre contexte *Navigation en milieu humain encombré*, nous privilégions des capteurs qui permettent d'acquérir des informations riches sur le monde 3D, possibles à traiter dans le systeme embarqué sur le robot ; ces capteurs doivent être robustes vis-à-vis des chocs ou de gestes malveillants d'utilisateurs, donc plutôt compacts ; ils ne doivent présenter aucun risque pour l'Homme, donc plutôt passifs ; ils doivent pouvoir être produit à bas coût. Le capteur privilégié est la caméra ; c'est un capteur passif, sans risque pour la santé de l'Homme, sans interférence possible avec d'autres

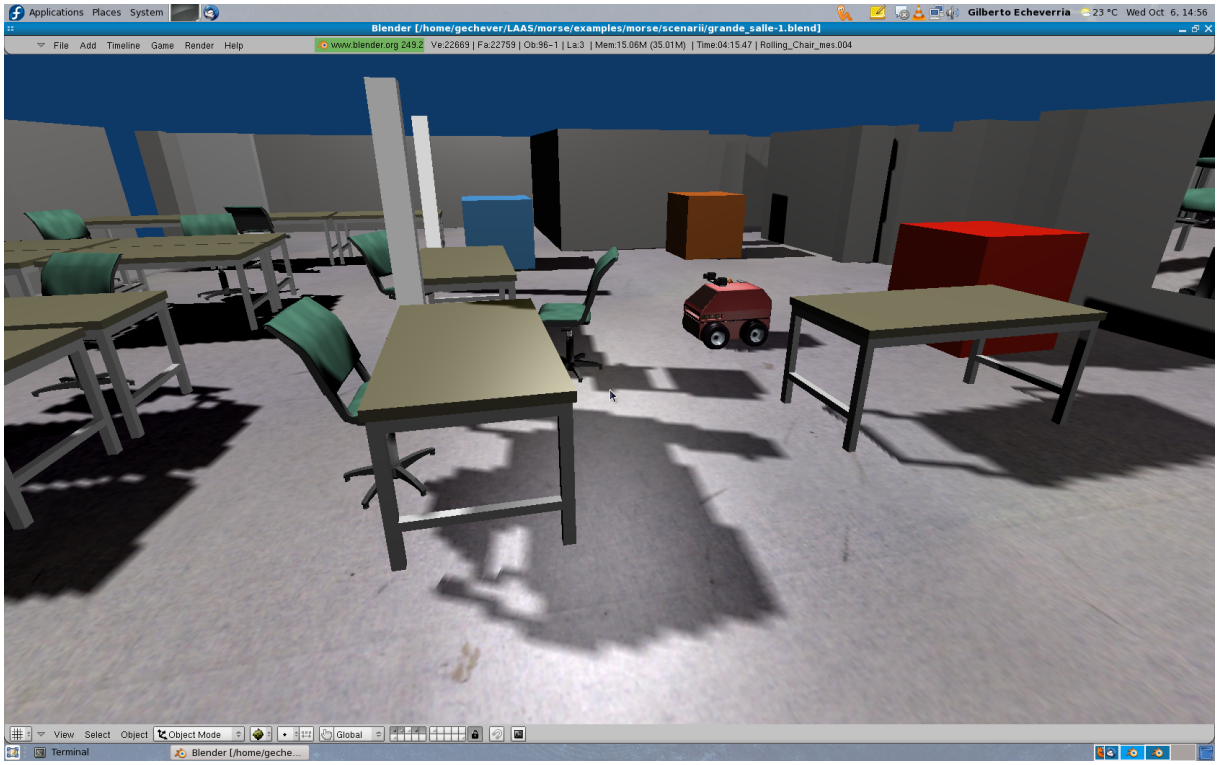


FIGURE 1.1 – Simulation d'un environnement de navigation pour un robot en milieu intérieur.

capteurs, qui fournit beaucoup d'informations sur l'environnement, à la fois sur la géométrie, l'apparence et le mouvement des objets proches du robot. La diffusion massive de caméras dans les téléphones ou ordinateurs portables ont permis aux industriels de développer des micro-caméras de bonne qualité à très bas coût.

1.1.2 Vision et système embarqué

Cependant, de la même manière que chez l'humain, la vision est le sens qui transmet le plus d'informations au cerveau, les caméras transmettent des images de grande résolution (au moins 640x480 couleur, si ce n'est 1MB) et à haute fréquence (typiquement 30Hz), qu'il faut analyser par des calculs très répétitifs. Les capacités de calcul embarquées sont donc un autre point clé pour exploiter massivement des caméras sur des robots de service. En plus d'être gourmandes en puissance de calcul, les applications robotiques imposent aussi des contraintes de temps qu'il convient de respecter, notamment pour limiter le temps de réaction du robot en cas de détection d'un événement non prévu. Les performances requises pour ces applications mettent à mal les systèmes informatiques classiques et cela s'accentue encore avec les contraintes d'espace ou de consommation.

Un système embarqué sur un robot ou un véhicule, est soumis à de fortes contraintes de temps de calcul, d'espace et de consommation d'énergie. Il convient souvent de développer un système de calcul dédié à l'application. Un système embarqué est composé d'unités de calcul, de mémoires pour stocker les données, de périphériques, etc. Ces composants doivent être exploités de manière à tirer le maximum pour l'exécution de l'application ; mais avec l'évolution des exigences de puissance de calcul, il faut accroître les capacités de calcul et de mémoire dans les systèmes embarqués, exploiter le parallélisme des opérations en multipliant les unités de calcul. Cette évolution conduit à une augmentation de la complexité des systèmes ; le temps de conception, le coût des composants et la consommation d'énergie

deviennent des facteurs importants.

Différents systèmes embarqués de vision en temps réel ont été mis en œuvre sur des architectures dédiées (ASIC, FPGA, etc.) récemment. Les solutions proposées permettent d'exécuter des opérations en parallèle afin de répondre aux contraintes de la performance, mais ainsi de minimiser la consommation d'énergie et/ou le coût par unité du système. Néanmoins, le problème majeur avec ces implémentations sur matériel est le temps, les coûts requis pour concevoir et pour préparer la fabrication, en sus des coûts non recouvrables de la production. Ainsi, les systèmes embarqués de vision sont souvent développés soit pour les applications de niche pour lesquelles des coûts élevés du matériel ou du développement sont tolérés (exploration spatiale, défense,etc.), soit au contraire, pour des systèmes qui sont produites en quantités massives, où le coût de développement est absorbé par la baisse du coût unitaire (applications grand public sur les portables, vidéo-surveillance,etc.).

Les Field-Programmable Gate Array (FPGA) ont permis la création de systèmes embarqués standard, produits à grande quantité, ce qui amortit le coût et réduit considérablement le temps d'accès au marché pour les solutions matérielles. Cependant, le coût d'ingénierie et la conception de solutions à base de FPGA demeurent nettement plus élevés que pour les solutions logicielles. Les concepteurs doivent souvent boucler sur le processus de conception afin de répondre aux contraintes de performances du système tout en minimisant la taille requise du FPGA. Chaque itération de ce processus prend des heures ou des jours.

Traditionnellement, la conception de systèmes FPGA a été développée par des ingénieurs électroniciens spécialistes d'un langage de description matérielle. Bien que de nombreux langages aient été proposées, Verilog et VHDL se sont imposés pour la conception de systèmes sur FPGA. Ces langages nécessitent une connaissance large et spécialisée des techniques de conception électronique pour obtenir la performance requise dans l'application, de sorte que, ces langages ne sont pas faciles à manipuler. En outre, si l'algorithme est complexe et/ou les contraintes de l'application sont difficiles, le développement de l'application peut prendre plusieurs années. Alors, au cours des dernières années, plusieurs approches ou produits ont été proposés pour la conversion de descriptions d'algorithmes en langage de haut niveau (comme le langage C) vers des descriptions d'implantation sur matériel. Ce type d'approches est appelée la synthèse de haut niveau et de nombreux outils peuvent être utilisés : Catapult-C, HandelC, Impulse C, GAUT, ROCCC, SPARK, SYNDEX entre autres.

1.2 Problématique

L'émergence du nouveau marché de la robotique de service dans des environnements humains, comme les robots personnels à domicile, les transporteurs de bagages dans une station de transport, les chariots de supermarché dans un centre commercial, etc, va rendre obligatoire l'intégration de technologies bon marché, efficaces et sûres sur le robot, notamment afin de détecter les obstacles lors des mouvements. Nous avons participé au projet européen FP6 CommRob, projet dédié au développement d'un trolley robotisé mis à disposition du public dans un centre commercial ; la figure 1.2 présente un des démonstrateurs intégrés lors de ce projet.

Traditionnellement pour les robots en milieu intérieur, la détection d'obstacles est faite par des capteurs à courte portée tels que des capteurs à ultrasons ou infrarouge ; ces capteurs sont montés comme une ceinture tout autour du robot. On sait que ces capteurs acquièrent des données très clairsemées et pauvres, et que les analyses basées sur ces données sont peu fiables : de nombreux faux positifs -faux obstacles détectés- et surtout, faux négatifs –vrais obstacles non détectés. Néanmoins, parce qu'ils sont très compacts, de très faible coût et qu'ils exigent très peu de calcul, ces capteurs à ultrasons sont maintenant massivement intégrés sur les véhicules comme capteurs de recul pour le stationnement, mais seulement afin d'aider au conducteur. Sur un robot autonome, la fiabilité de la détection d'obstacles est améliorée en

FIGURE 1.2 – Le démonstrateur de *Shopping Trolley*.

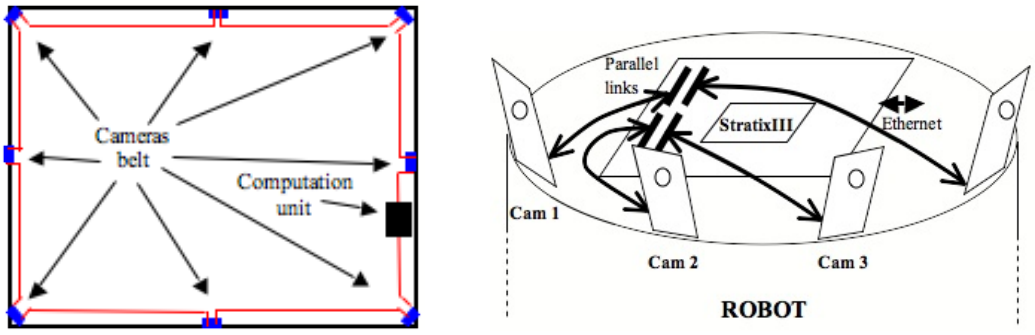
utilisant une grille d'occupation probabiliste, construite en ligne afin de fusionner les données acquises au cours du mouvement.

Beaucoup de robots en milieu intérieur sont également équipés de télémètres laser tels que les capteurs SICK, IBEO ou HOKUYO en tant que *safety sensor* ; ces capteurs fournissent en haute fréquence, la distance aux surfaces détectées dans un plan horizontal parallèle au sol. Ces données sensorielles sont exploitées pour la détection d'obstacles, mais ces obstacles doivent couper le plan du laser, de sorte que les formes proéminentes au-dessus ou au-dessous du plan laser ne sont pas détectées. Pour les applications dans un lieu public, avec présence possible d'enfants, de nombreux inconvénients apparaissent : sécurité oculaire, coût et difficulté d'identifier les objets à partir de seulement quelques points détectés sur les jambes d'une personne ou sur les côtés d'un autre robot.

1.3 Notre approche

Nous proposons l'utilisation d'un système de vision multi-caméras pour détecter les obstacles lors de la navigation du robot. Pour un robot de service qui se déplace dans un environnement humain, la vision a de nombreux avantages par rapport aux autres capteurs : la sécurité des yeux (détection passive), le coût (avec la large diffusion des caméras dans les téléphones portables, le coût des micro-caméras est de quelques euros), la faible consommation d'énergie de la caméra elle-même, la compacité (une micro-caméra peut occuper moins d'un centimètre-cube), la richesse des données sensorielles acquises (haute résolution, couleur, aspect, etc.). Mais il est bien connu que la vision n'est pas seulement un capteur d'orientation : le traitement de l'image est très sensible aux conditions du milieu (variations d'éclairage, occultation, etc.) et l'analyse des scènes à partir de données visuelles est un processus complexe, qui nécessite beaucoup de ressources de calcul et de mémoire, en particulier lors de la prise en compte des analyses spatio-temporelles (multi-caméras ou suivi monoculaire, traitement sur des séquences d'images,etc.).

En comparant à la ceinture classique de capteurs ultrasons, notre défi consiste à remplacer tous les capteurs de proximité par une micro-caméra. Notre démonstrateur robot (voir figure 1.3) a un champ de



(a) (b)

FIGURE 1.3 – A gauche, une ceinture de micro-caméras sur un robot ; à droite, système réalisé avec uniquement 4 caméras

vue de 150° dans la version courante, mais il sera omnidirectionnel dans la version finale : ainsi afin de détecter les obstacles, quelle que soit l'orientation de mouvement, les données sensorielles seront acquises par N caméras (4 dans la version courante, 8 dans la version finale). Comme le nombre de caméras est important, cela rend obligatoire la conception et la mise en œuvre d'une architecture dédiée pour le traitement des images. L'objectif est de construire et mettre à jour une grille d'occupation robot-centrée lors de la navigation du robot. Cette opération doit être exécutée à 30 Hz, ce qui réduit la latence entre l'acquisition de l'image et la mise à jour de la grille. Cette grille sera envoyée au module de commande du robot.

(a) (b)

FIGURE 1.4 – Ground image : examples.

Afin de faire la détection des obstacles sur chaque image, nous avons essentiellement développé une approche basée sur l'apparence ; en effet la figure 1.4 montre que le sol est généralement de couleur et texture uniformes. D'abord, chaque pixel est transformé en l'espace de couleur CIE-Lab qui a la plus grande immunité aux changements d'éclairage. En chaque pixel, les composantes chromatiques a et b sont utilisées comme attributs de couleur. La composante L est utilisée pour calculer les attributs de texture. Les attributs de texture sont obtenus en utilisant la technique des histogrammes de sommes et de différences. Cette technique a été optimisée afin de réduire les ressources nécessaires à sa mise en œuvre, en particulier pour éviter le calcul des histogrammes, ce qui réduit ainsi la mémoire requise et le nombre d'opérations. Avec cette technique, les attributs de texture sont calculés directement à partir des images de sommes et de différences de luminosité. Les attributs de couleur et texture sont utilisés pour

classer chaque pixel, afin qu'on puisse savoir si le pixel appartient ou pas à une classe particulière, et ainsi on peut déterminer si ce pixel est l'image d'un point de la scène qui appartient ou non à un obstacle. Le module de classification est fondé sur la méthode AdaBoost ; les paramètres de la classification sont automatiquement générés hors ligne à partir d'exemples choisis par un expert.

En raison des contraintes de l'application et du type de traitement, différentes architectures sur FPGA ont été développées. Le but est de tirer avantage de l'algorithme et de son traitement en parallèle, afin d'obtenir une performance supérieure. Notre système dans son état actuel, est composé de trois modules principaux, avec trois architectures conçues pour la transformation de couleur, pour l'analyse de texture et pour la classification basée sur l'apparence. Nous avons proposé une méthode pour régler automatiquement les paramètres de cette classification, cela pour réduire les erreurs humaines et optimiser le temps de conception.

1.4 Objectifs et Contributions

Notre objectif est donc de concevoir, développer, implémenter, et valider un système de *«Vision multi-caméras pour la détection d'obstacles sur un robot de service : des algorithmes à un système intégré»*.

Pour répondre aux contraintes de l'application de détection d'obstacles pour la navigation d'un robot dans un milieu humain, nous nous sommes fixer les objectifs suivants :

- Développer et valider une méthode visuelle permettant de détecter les obstacles par leur apparence dans un environnement humain.
- Développer et valider un algorithme d'analyse des attributs de texture qui permet son implémentation sur des systèmes parallèles.
- Développer et valider une méthode pour automatiser la création des modules de classification fondée sur l'algorithme AdaBoost.
- Concevoir et implémenter une architecture sur FPGA qui permet de faire l'analyse dense des attributs de texture en temps réel.
- Concevoir et implémenter une architecture sur FPGA qui permet de faire la classification de tous les pixels d'une image en temps réel pour la détection d'obstacles.
- Intégrer sur un démonstrateur tout ou parlie des modules développés, et valider ce système par des expérimentations de navigation en milieu humain.

Nos contributions sont donc les suivantes :

- Une méthode de détection d'obstacles basée sur les attributs de couleur et texture. Cette méthode exploite une base d'apprentissage pour classifier les objets perçus dans l'environnement de navigation du robot.
- L'analyse de l'adéquation entre l'algorithme d'analyse de texture basé sur les histogrammes des sommes et des différentes et une architecture matérielle dédiée au calcul de ces attributs. Cette adéquation permet d'optimiser les ressources requises pour son implémentation et d'augmenter ainsi la performance en temps réel.
- Une méthode pour la génération du module de classification des objets depuis les images acquises sur l'environnement du robot. Les différents outils requis pour l'implémentation de cette méthode, ont été développés afin d'accélérer le processus de conception et de configuration du module de classification.
- La réalisation et la validation partielle d'un système de vision multi-caméras pour la détection d'obstacles intégré sur un robot de service.

Une partie du travail décrit dans ce manuscrit a été publié dans des conférences internationales :

- **Mario-Alberto Ibarra-Manzano**, Michel Devy, Jean-Louis Boizard, "Real-Time Classification Based on Color and Texture Attributes on an FPGA-Based Architecture", *Conference on Design and Architectures for Signal and Image Processing, DASIP 2010*, Edinburgh, Scotland, ECSI - Electronic Chip and Systems design Initiative, du 26 au 28 Octobre 2010, pp. 53-59.
- **Mario-Alberto Ibarra-Manzano**, Michel Devy, Jean-Louis Boizard, Pierre Lacroix, and Jean-Yves Fourniols, "An efficient reconfigurable architecture to implement dense stereo vision algorithm using high-level synthesis", *19th International Conference on Field Programmable Logic and Applications, FPL 2009*, Prague, Czech Republic, IEEE Circuits and Systems Society, du 31 Août au 2 Septembre 2009, ISBN : 978-1-4244-3892-1, ISSN : 1946-1488, pp. 444-447.
- **Mario-Alberto Ibarra-Manzano**, Dora-Luz Almanza-Ojeda, Michel Devy, Jean-Louis Boizard, and Jean-Yves Fourniols, "Stereo vision algorithm implementation in FPGA using census transform for effective resource optimization", *12th EUROMICRO Conference on Digital System Design : Architecture, Methods and Tools, DSD 2009*, Patras, Greece, IEEE Computer Society, Los Alamitos, California, U. S. A., du 27 au 29 Août 2009, ISBN : 978-0-7695-3782-5, pp. 799-805.
- **Mario A. Ibarra-Manzano**, Jean-Louis Boizard, Michel Devy and Jean-Yves Fourniols, "Towards a reconfigurable obstacle detection system"', *Advanced Computer Architecture and Compilation for Embedded Systems, ACACES 2009*, Terrassa, España, High-Performance Embedded Architecture and Compilation (HiPEAC) Network of Excellence, Academia Press, Ghent, le 15 Juillet 2009, ISBN : 978-90-382-1467-2, pp. 19-22.
- **M. A. Ibarra-Manzano**, M. Devy, J. L. Boizard, P. Lacroix, W. Filali, J. Y. Fourniols, "Obstacle avoidance by a multi-camera system", *9th. International Workshop on Electronics, Control, Modelling, Measurement and Signals*, Mondragon, España, MU. Mondragon Unibertsitateko Zerbitzu Editoriala, du 8 au 10 Juillet 2009, ISBN : 978-84-608-0941-8, pp. 81-87.
- M. Devy, **M. Ibarra Manzano**, J.L. Boizard, P. Lacroix, W. Filali and J.Y. Fourniols, "Integrated subsystem for Obstacle detection from a belt of micro-cameras", *14th International Conference on Advanced Robotics, ICAR 2009*, Munich, Germany, German Robotics Society, IEEE Robotics and Automation Society, GPS Gesellschaft fur Produktionssysteme GmbH, du 22 au 26 Juillet 2009, ISBN : 978-1-4244-4855-5, pp. 887-892.

1.5 Organisation du manuscrit

Nous travaux fournissent des éléments de réponses aux différentes interrogations suivantes :

Chapitre 2 : La détection d'obstacles. Ce chapitre propose une étude bibliographique brève des solutions proposées pour la détection d'obstacles dans le contexte de navigation en intérieur et en extérieur. Nous présenterons différents types de capteurs actifs et passifs et les solutions adaptées à chacun, avec notamment les algorithmes de détection d'obstacles par vision monoculaire et stéréo.

Chapitre 3 : Algorithmes pour la détection d'obstacles. Nous présentons l'algorithme global pour la détection des obstacles, algorithme composé de deux méthodes de détection indépendantes. La première méthode utilise les attributs de couleur et texture pour différencier le sol et les obstacles. Nous introduisons une adéquation algorithmique qui va nous permettre d'optimiser le calcul des attributs de texture. Puis, nous comparons différents algorithmes de classification pour sélectionner celui qui a la meilleure performance avec les attributs de couleur et ltexture utilisés. Ensuite, nous présentons une analyse des divers paramètres, pour comprendre les effets de chaque paramètre dans la performance de la classification. Dans la deuxième partie de ce chapitre, nous présentons la deuxième méthode de détection des obstacles qui utilise une approche géométrique, les bases théoriques et les résultats sont également présentés. Enfin, nous proposons de fusionner toutes les détections par la construction d'une grille d'oc-

cupation, ce qui permet d'exprimer la position des obstacles détectés dans chaque image, dans un repère fixe lié au plan du sol. Le système transmet cette représentation aux autres fonctions en charge de la navigation du robot.

Chapitre 4 : Méthodologie et outils pour le développement d'architecture matérielle. Nous présentons le positionnement technologique de la logique programmable, puis la méthode de conception dite *classique* fondée sur des langages de description du matériel. Après cela, nous présentons la synthèse de haut niveau comme une solution pour l'accélération de la conception d'architectures sur matériel. Enfin, une comparaison des deux méthodes est effectuée pour l'algorithme de stéréovision.

Chapitre 5 : Architecture matérielle pour la détection d'obstacles. Ce chapitre propose des architectures pour la transformation de l'espace de couleur, l'analyse des attributs de texture et l'algorithme de classification. Dans ce chapitre aussi nous présentons une analyse comparative avec les différentes architectures trouvées dans la littérature qui font un traitement similaire. Notre démonstrateur du système intégré de vision multi-caméras est décrit à la fin de ce chapitre.

Conclusion générale et perpectives. Nous concluons par le bilan des travaux effectués et détaillons les contributions apportées avant d'aborder quelques perspectives à nos travaux.

Chapitre 2

La détection d'obstacles

Sommaire

2.1 Introduction

Les systèmes de navigation autonome, actuellement, privilégient des capteurs fournissant des informations sur les caractéristiques de l'environnement, comme des systèmes mono ou multi-caméras, les télémètres laser, les radars, des sonars parmi d'autres. Chaque capteur a des avantages et des inconvénients propres ; dans le contexte du développement de robots de services destinées au grand public, la tendance actuelle est de privilégier les méthodes de perception fondées sur des capteurs compacts (donc faciles à intégrer dans la structure mécanique du robot), sans danger pour l'utilisateur et bas coût. Ce sont les capteurs visuels qui satisfont le mieux ces contraintes, mais selon l'application finale, la vision n'est pas toujours la solution la plus performante.

Dans ce chapitre nous allons présenter les méthodes existantes pour la détection des obstacles depuis un véhicule ou un robot : ces méthodes appliquent des algorithmes qui exploitent des données sensorielles acquises depuis un ou plusieurs capteurs. D'abord, la section 2.2 présente les solutions les plus utilisées, fondées sur les capteurs actifs. Puis la section 2.3 décrit quelques travaux dédiées à la détection d'obstacles par des algorithmes de vision.

2.2 Détection par des capteurs actifs

Un capteur est dit "actif" s'il produit une énergie envoyée sur la scène ; les obstacles sont détectés par l'énergie réfléchie par leur surface. Un capteur actif possède donc à la fois un émetteur et un récepteur.

Nous décrivons rapidement les méthodes fondées sur la télémétrie laser et le radar. Ces deux capteurs actifs sont les plus utilisées sur des véhicules ou sur des robots d'extérieur, parce qu'ils sont robustes aux variations des conditions atmosphériques (pluie, neige) ou à des conditions dégradées d'illumination ; les méthodes visuelles connaissent beaucoup de défaillances dans ces situations.

Sur les robots de service évoluant en milieu intérieur, les capteurs ultrasonores ou infrarouges sont également très souvent exploités pour détecter des obstacles proches pendant les mouvements ; ils sont intégrés sous la forme de ceintures de capteurs disposés tout autour du robot. Depuis quelques années, des proximètres ultra-sons sont également intégrés dans les pare-chocs de véhicules grand public, en tant que capteur de recul. Nous ne détaillerons pas ces méthodes, considérées généralement comme peu fiables, peu précises et très pauvres dans la communauté robotique.

2.2.1 Détection par télémètre Laser

La télémétrie laser combine un système électronique pour la mesure ponctuelle de distance à partir d'un faisceau laser réfléchi par l'obstacle, et un mécanisme de balayage pour acquérir des mesures de distance dans une région d'intérêt. La lumière réfléchie par un obstacle quelconque se trouvant dans l'axe du tir revient vers le capteur et est perçue par un récepteur ; la distance capteur-obstacle est obtenue de manière classique par le temps de vol d'une impulsion laser, ou par le déphasage d'un signal modulé en amplitude ou en fréquence. Tous les impacts laser sont dans un plan si le balayage est seulement en azimut, ou sont dans l'espace 3D si le balayage est en site et azimut.

L'emploi de la télémétrie laser est très courant sur des véhicules haut de gamme, ou sur des robots de service, car la mesure est rapide, la portée peut être importante (plusieurs centaines de mètres selon le type de scanner utilisé : voir figure 2.1) et les taux d'échecs (faux négatifs ou faux positifs) très faibles. Citons par exemple [43] où les auteurs utilisent les mesures acquises par un scanner laser IBEO pour détecter les objets dans la scène. Ce capteur retourne une image de profondeur haute résolution. Dans [86] le module de détection d'obstacles emploie un filtre d'association probabiliste et des réseaux bayésiens pour estimer la probabilité de détection à l'instant suivant.

(a) (b)

FIGURE 2.1 – a) Scanner laser de haute résolution de la compagnie Ibeo. b) Laser LMS200 de Sick Inc. en Allemagne, ce capteur est fréquemment utilisé sur les robots de service pour contrôler l'exécution de mouvements à vitesse modérée.

Cependant, il arrive fréquemment que les systèmes robotiques soient équipés avec des télémètres laser de plus basse résolution, comme dans les travaux de Mendes [108]. Pour ces télémètres, l'information

est parfois insuffisante et même de légères vibrations peuvent perturber les mesures.

Ces capteurs laser sont donc excellents pour la détection des obstacles, mais leur faible résolution rend difficile l'identification des objets détectés. Pour pallier ce défaut, d'autres solutions ont été envisagées, en particulier la fusion des données laser avec l'information visuelle. Par exemple, pour des applications dans la sécurité routière, Labayrade and al [92] utilise simultanément du laser et la stéréovision. Par ailleurs, on trouve dans la littérature de très nombreux travaux, qui pour différents contextes (navigation autonome d'un robot en extérieur et en intérieur, détection de piétons depuis un véhicule intelligent) proposent de fusionner les données d'un télémètre laser 2D avec un système visuel monoculaire [141], [88] et [49] ; généralement le laser permet la détection, la vision permet le suivi et l'identification.

2.2.2 Détection par radar

Le radar (RAdio Detection And Ranging) consiste à mesurer les signaux de radio réfléchis en exploitant la théorie de Doppler pour détecter le déplacement en fréquence des ondes réfléchies. Le radar travaille à très grandes portées ; il peut être utilisé en statique (par exemple, détection d'intrusion dans un environnement sensible) ou mobile (par exemple, détection d'obstacles depuis un véhicule). Cependant, comme pour les signaux lumineux émis par un télémètre laser, la portée de détection maximale d'une cible réfléchissant des signaux hyperfréquences, dépend de la texture de la cible et de la spécularité, ou angle entre rayon incident et normale à la surface. Concernant les applications Robotique, le radar est resté longtemps peu exploité à cause de son coût, et surtout, de la difficulté pour acquérir un capteur, car cette technologie était principalement réservé aux applications de défense ; cette situation a changé depuis l'introduction du radar dans les ACC des véhicules intelligents, et on voit se multiplier les travaux sur le radar dans la communauté robotique (citons le projet IMPALA en France à Clermont-Ferrand, les travaux de M.D.Adams à Singapour, etc.).

La résolution angulaire d'un capteur radar est d'une vingtaine de degrés, mais le faisceau du radar, lors de sa propagation, peut détecter plusieurs cibles en même temps dans une région d'intérêt, car tous les échos successifs sont détectés. Cette caractéristique est exploitée dans [97] pour la détection de piétons dans un milieu urbain. Le système présenté dans ce travail est constitué de deux capteurs radar de courte portée situés à l'avant d'un véhicule expérimental de Daimler, avec une portée de 30m environ. La difficulté du système est d'identifier les échos renvoyés par des piétons ou par d'autres objets. Pour ce faire, les auteurs proposent une fusion de données entre le radar et des capteurs thermiques ; ces derniers servent à détecter la radiation thermique émise par les humains. La fusion des données utilise une approche probabiliste, pour construire une grille d'occupation centrée sur la position du robot.

Bien que le radar ne soit pas affecté par la pluie ou la neige, il est très perturbé par des sources d'interférence magnétique. De plus, la facilité avec laquelle un radar détecte un objet à une distance donnée, est directement liée à sa surface équivalente radar (SER). Un véhicule par exemple à une SER ($10m^2$) supérieure à celle d'un piéton ($2m^2$) [93]. La plupart des petits obstacles ont une SER encore plus faible ce qui les rend indétectables. Cet inconvénient rend indispensable l'association d'autres capteurs avec le radar. Le ladar (association laser et radar) a été proposé pendant les années 70 et 80 [106] grâce à sa meilleure vitesse d'opération par rapport aux capteurs de vision. Cependant dans les années 90, l'évolution des systèmes informatiques a rendu possible, l'utilisation des algorithmes de vision en temps réel : à partir de ce moment-là, il a été possible d'intégrer système radar ou ladar avec des systèmes visuels mono ou multi-caméra pour détecter les objets pendant la navigation d'un véhicule intelligent [90].

2.2.3 Conclusion pour les capteurs actifs

Les technologies laser et radar fournissent des données sensorielles 3D en temps réel, à haute fréquence ; l'exploitation de telles données rend plus robuste la tâche de détection et localisation d'objets par un système robotique mobile ou véhicule intelligent, même dans des environnements complexes comme dans des scènes urbaines ou routières. Par contre, s'il y a plusieurs systèmes autonomes qui, en même temps, utilisent ces capteurs actifs, l'interférence entre eux sera inévitable. Soulignons aussi qu'un système ladar a un coût élevé, ce qui doit être considéré avant sa mise en œuvre dans un projet. D'autre part un laser scanner est fortement perturbée dans ces mesures par l'attitude du véhicule sur lequel il est monté, en particulier par l'angle de roulis (rotation autour de l'axe longitudinal du véhicule).

L. Matthies et. al. [107] indiquent que le choix entre radar, laser ou vision pour détecter des obstacles dans une scène depuis un véhicule, dépend de la fréquence d'apparition des objets dans l'environnement de navigation et de la vitesse du véhicule qui porte les capteurs. Si la fréquence est faible et les vitesses aussi, il est plus avantageux d'exploiter la vision. Par exemple, les auteurs précisent que l'utilisation d'un système ladar pour un robot d'exploration planétaire, n'est pas nécessaire vu la très faible fréquence d'apparition d'obstacles et vu la faible vitesse du véhicule.

2.3 Détection d'objets par des capteurs visuels passifs

Un capteur passif ne fait que recevoir l'énergie naturellement réfléchie par les objets présents dans la scène. Plusieurs capteurs passifs sont utilisés pour la robotique ou pour la surveillance : les capteurs inertiels (détection de vibrations, de mouvement, etc.), les capteurs haptiques (contact), mais les plus connus sont les capteurs CCD ou CMOS, donc des caméras [126]. En tant que capteur passif, une caméra mesure la luminosité de l'environnement sous la forme d'une image digitalisée, ou d'un tableau de pixels. Cette section étudie les deux configurations essentielles utilisées pour acquérir ces images : les systèmes stéréo et monoculaire, ainsi que les différentes manières d'analyser ces images. Nous commencerons par rappeler les concepts fondamentaux de la stéréovision avant de décrire les principales méthodes proposées pour détecter des obstacles mobiles à partir de données sensorielles acquises par stéréovision. Ensuite, nous décrirons l'analyse spatio-temporelle réalisée sur les images monoculaires dans le but de décrire son contenu. Finalement, une brève conclusion à propos des capteurs caméra sera présentée à la fin de cette section.

2.3.1 La stéréovision

Les systèmes de stéréovision permettent de calculer par triangulation, la profondeur des objets dans l'espace 3D, grâce à leurs projections dans les deux caméras. La différence des positions dans les deux images, de la projection d'un même point de l'espace 3D, s'appelle la disparité ; la recherche des points correspondants entre ces deux images, permet donc de produire une image de disparité. En supposant que les caméras ont des axes optiques parallèles, comme dans le capteur Bumble Bee (figure 2.2b), alors la disparité est inversement proportionnelle à la profondeur ; ainsi la disparité est nulle pour un point à l'infini, typiquement un point sur la ligne d'horizon.

La figure 2.2 illustre des montages typiques de cameras en configuration stéréo, connu aussi comme banc stéréo. Si le banc stéréo est calibré, les coordonnées du point 3D correspondant à ce pixel peuvent être calculées. On sait que la précision du capteur stéréo est proportionnelle à la profondeur au carré ; de ce fait, la portée utile d'un tel capteur est limitée, typiquement quelques mètres sur des focales courtes (5m. par exemple pour la détection d'obstacles proches en site urbain), quelques dizaines de m. sur des focales longues (40 ou 50m. max pour la détection d'obstacles lointains sur route).

(a)

(b)

FIGURE 2.2 – (a) Configuration stéréo canonique sur notre système stéréo du prototype PICAS$O (Plateforme d'Intégration de CApteurs multi-Sen$Oriels), avec deux caméras dans le visible entourant une caméra thermique, (b) Système de stéréovision Bumble Bee2 fabriqué par Point Grey Research©.

Les approches pour construire l'image de disparité depuis une paire de caméras sont classifiées en méthodes éparses et denses. Dans les méthodes éparses les images droite et gauche sont d'abord prétraitées pour extraire des indices visuels discriminants (points d'intérêt, segments, etc.). Ce sont ces indices qui sont appariés et reconstruits. Dans le cas des méthodes denses, les pixels sont tous appariés en exploitant une mesure de ressemblance (corrélation par SAD, SSD, ZNCC, etc., distance de Hamming entre code CENSUS). Ces méthodes denses peuvent être optimisées en adoptant une approche pyramidale : les disparités sont d'abord recherchées à faible résolution (images 80×60) avant d'être raffinées en descendant la pyramide jusqu'à l'image initiale (640×480 par exemple).

Le problème classique de la vision, est le temps de traitement des images, notamment pour la détection d'obstacles qui requiert une grande réactivité. Plusieurs solutions ont donc été proposées pour diminuer le temps de calcul de l'image de disparité, en particulier en employant les systèmes re-configurables principalement les FPGA (Field Programming Gate Array). L'avantage de l'implémentation d'un algorithme sur FPGA, réside en sa capacité d'exécuter plusieurs opérations en parallèle, ce qui accélère considérablement le calcul même pour un processus dense. Un algorithme de stéréovision fondé sur la transformée CENSUS et la mise en correspondance par distance de Hamming, a été développé dans notre équipe sur une architecture FPGA par Boizard et al. [14]. L'architecture proposée permet de génèrer une image de disparité dense à 100Hz pour la résolution 640x480 pixels. La navigation d'un robot mobile ou d'un véhicule autonome fondée sur la vision stéréo, peut donc être effectuée en temps réel, même si les conditions de l'environnement ne sont pas très favorables.

Le problème de l'auto-calibration

Le besoin de calibrer un banc stéréo (comme celui montré en figure 2.2a) périodiquement après son montage sur un robot ou un véhicule, est considéré comme un des principaux problèmes liés à la stéréovision ; en effet, la performance d'un banc stéréo (nombre de pixels appariés, nombre de faux appariements, précision de la reconstruction 3D, etc.) dépend totalement du calibrage. Cette tâche est toujours un sujet de recherche très étudié, et cela depuis 1992 (date de parution de l'article fondateur de O.Faugeras). C'est un problème difficile, surtout quand il faut le résoudre en temps réel (donc sans faire appel à des méthodes d'optimisation trop lourdes).

Vu cette limitation, au LAAS il a été proposé dans les travaux de J.Sola [129], une approche limitée de correction du calibrage en considérant connus les paramètres intrinsèques des caméras. L'auteur propose

de ré estimer la rotation entre les deux caméras dans la boucle de localisation sur la bae d'amers visuels appris soit au préalable soit en simultanée (cas du SLAM). Cependant, aujourd'hui il est possible de se passer de cette étape, car sur le marché, il existe des bancs stéréo pré calibrés, comme le BumbleBee2 de Point Grey Research©(voir photo en figure 2.2b). Par ailleurs ce système stéréo produit une image de disparité à la résolution VGA à plus de 48Hz ; même si ces images de disparité sont assez imprécises et contiennent quelques artefacts, les grands problèmes de la stéréovision sont bien résolus par ce système. Le bémol identifié pour ce banc stéréo est qu'il continue à être cher, que sa configuration est rigide (base fixe) et surtout que l'utilisateur dépend directement de techniques préprogrammées pour l'obtention de la carte de disparité.

La détection d'obstacles par stéréovision

Citons les nombreux travaux du JPL en Californie sur la détection d'obstacles mobiles depuis un capteur stéréo embarqué sur un robot. Dans certains cas, ils estiment l'ego- motion du robot pour distinguer le mouvement apparent des composantes statiques de la scène de celui des composantes mobiles. Cette estimation est ainsi exploitée pour corriger le flot optique en "retirant" le mouvement du système, comme cela est proposé dans [133]. Dans ce travail, Talukder et al. utilisent des champs denses de disparité stéréo et ils suppriment les petites régions non connectées dans le champ du flot estimé.

D'autre part, si on considère résolu le problème de la calibration dans le cas de la navigation, alors la détection d'obstacles peut être réalisé par une rectification homographique [142]. La rectification homographique transforme une des images de la paire stéréo (par exemple, la droite), de manière à ce que le plan de la route soit identique entre image gauche et image droite rectifiée. Une simple différence entre ces deux images permet alors de discriminer les pixels correspondants à des points 3D qui ne sont pas portés par ce plan. Après que ces pixels associés à des points 3D au dessus ou en dessous de la route aient été mis en évidence, les obstacles sont segmentés, puis caractérisés par une analyse de connexité et de similarité. La rectification homographique est appliquée a priori à l'aide d'une calibration préalable du système stéréo et du plan de la route.

Signalons deux approches comparées par V. Lemonde [96] au LAAS-CNRS pour la détection d'obstacles proches (à moins de 5m. par exemple), à partir de la stéréovision : on peut faire sans risque dans ce cas, une hypothèse de route plane. D'abord l'auteur a exploité le concept de v-disparité, qui permet de détecter les obstacles directement à partir de l'image de disparité. Depuis les travaux de R.Labayrade [92], il est connu que cette approche est très sensible aux erreurs sur l'attitude du capteur vis-à-vis du plan de la route.

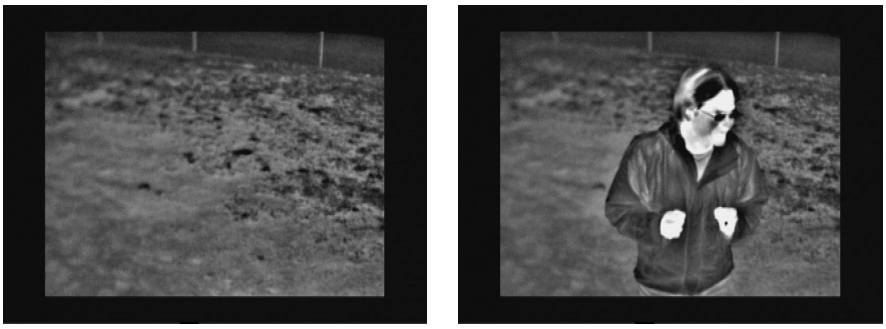
Puis, V.Lemonde a proposé de détecter les obstacles à partir de l'image de points 3D reconstruites à partir de l'image de disparité, simplement par seuillage sur l'élévation des points 3D au-dessus du plan de la route. Les points étiquetés *Obstacle* sont ensuite regroupés en fonction d'un critère de connexité. Finalement, la création d'enveloppes permet de modéliser ces ensembles de points par des parallélépipèdes rectangles. Ces boites englobantes permettent d'associer les obstacles détectés d'une image à la suivante, et d'estimer grossièrement leurs dynamiques s'il s'agit d'obstacle mobile. L'auteur propose ensuite une approche très inspirée de la v-disparité, sauf que les cumuls se font dans le 3D, donc successivement sur les coordonnées X,Y et Z des points de l'image 3D, ce qui rend plus facile le regroupement des points *Obstacle* en objets disjoints, et ce qui permet également de traiter de manière plus robuste, le cas du dévers de la route ou du roulis non nul du véhicule.

Ces méthodes supposent que la route est plane, ce qui en pratique, la rend inexploitable pour la navigation sur terrain accidenté en milieu naturel ; dans ce cas, la détection d'obstacles se mue en la construction d'une carte de traversabilité du terrain, à partir d'une carte d'élévation (ou Modèle Numérique du Terrain).

2.3.2 Capteur de vision Infrarouge (IR)

Malgré leur faible résolution et le mauvais contraste des images infrarouges, les caméras infrarouges sont exploitées pour la navigation de véhicules dans des conditions de visibilité difficile, soit la nuit, soit en cas du fait des conditions météo (brouillard, etc.). On trouve dans la littérature de nombreux travaux sur la détection d'objets depuis des caméras IR de nuit [106] ou avec de la fumée [51].

La bande de fréquence de l'infrarouge est divisée en 3 : bande I (1-1,7 mm), bande II (3-5 mm) et bande III (8-12 mm). Les caméras IR correspondantes ont différentes capacités et propriétés. Par exemple l'eau absorbe plus ou moins les ondes infrarouges selon la fréquence. En plus, les images infrarouges ne sont pas très texturées, ce qui empêche l'utilisation des techniques traditionnelles pour la détection de texture ou d'indices visuels. Plusieurs images acquises par une camera infrarouge [1] sont montrées dans la figure 2.3.



(a) (b)

FIGURE 2.3 – Exemple d'images acquises par une caméra Infrarouge Raytheon Thermal-eye 2000B.

En ce qui concerne la navigation d'un véhicule intelligent dans un milieu extérieur inconnu, citons les travaux de Talukder et al. [133] réalisés avec un véhicule expérimental équipé avec une paire de caméras IR qui opèrent dans la bande 3-5 mm. Les algorithmes de stéréovision et du flot optique dense exécutés à partir d'images infrarouges, sont utilisés pour la détection d'objets dynamiques dans l'environnement. Plus récemment, Alonzo Kelly et al. [90] et Larry Matthies et al. [107] ont présenté une synthèse des résultats obtenus en matière de navigation soit sur des terrains naturels couverts de végétations, soit pour l'exploration planétaire. Ils concluent que l'exploitation de capteurs infrarouges est essentielle pour l'exploration de terrain inconnu, notamment pour identifier la nature des obstacles (autre véhicule, buisson ou rocher par exemple).

D'autres applications plus spécifiques de l'imagerie infrarouge ont été présentées en 2008 par L. Gond et al. dans [51] pour la surveillance de foules dans un lieu public (gare du métro). Les auteurs utilisent un capteur infrarouge non refroidi dans la bande 8-12 mm, pour reconnaître des situations d'incendies ou d'attaques chimiques. Ils ont analysé la capacité d'acquisition de ces capteurs infrarouges par rapport aux conditions d'illumination : ils précisent que la vision dans cette bande infrarouge est sensible à certains facteurs comme la lumière du soleil sur les objets, la pluie ou la neige. Dans ce contexte de vidéo surveillance (caméras fixes), la détection de personnes sur les images infrarouges est faite en appliquant la méthode de soustraction du fond en deux temps. Après une première soustraction de fond, ils calculent l'offset entre la moyenne des modes du modèle SKDA (Sequential Kernel Density Approximation), et la moyenne des pixels classés comme appartenant au fond. Cet offset est utilisé lors d'une seconde soustraction de fond, pour obtenir une image nettement améliorée.

1. Ces images font partie de "The Marulan Dataset" [131] téléchargés du site web [64]

Les caméras infrarouges présentent donc un intérêt certain pour la détection d'obstacles : leurs coûts encore élevés empêchent leur diffusion massive sur les véhicules grand public, mais ils sont exploités, souvent en fusion avec des caméras dans le visible, dans des applications spécifiques (sécurité civile, défense, etc.). Signalons au LAAS, les travaux en cours sur le projet SART (Système d'Aide au Roulage Tout Temps), pour la navigation d'aéronefs dans les aéroports.

2.3.3 Vision mono-caméra

Lorsque qu'on ne dispose que d'une unique caméra, les méthodes de détection d'obstacles se fondent sur la reconnaissance de formes ou sur la vision dynamique. Néanmoins, les approches les plus avancées combinent ces deux méthodes :

- Les modèles peuvent aussi tenir compte des comportements dynamiques caractéristiques d'une classe d'objets (vitesse min et max, type de trajectoires, parmi d'autres)
- Ces modèles peuvent être partiellement acquis ou mis à jour en ligne.

Dans la suite, nous présentons une synthèse des travaux sur l'analyse des images monoculaires en vue de la recherche d'objets statiques et dynamiques. Nous ne prétendons pas être exhaustif mais nous souhaitons présenter les principales méthodes utilisées dans ce contexte, car la vision monoculaire fait aussi parti, de notre problématique de recherche, dans la mesure où notre configuration multi-caméras n'est pas du type stéréovision.

La détection d'objets depuis des images mono-caméra

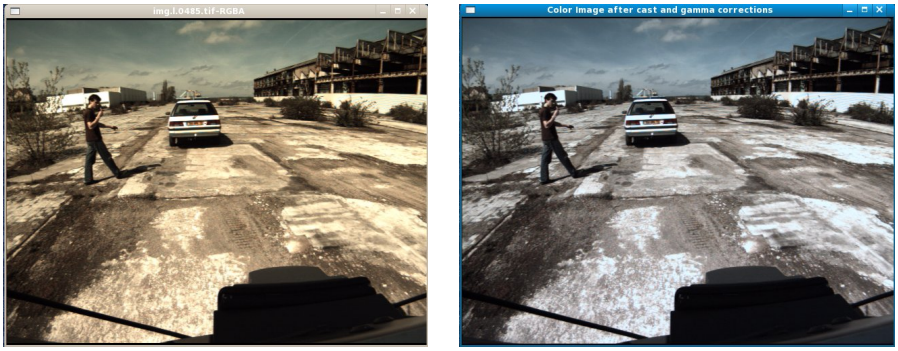
D'abord, nous trouvons dans la littérature que les techniques pour le traitement des images sont classifiées dans deux grandes approches : l'approche spatiale fondée sur l'extraction d'indices visuels dans une seule image, et l'approche temporelle fondée sur la détection de mouvement sur plusieurs images successives. Les approches spatio-temporelles traitent de la détection et du suivi d'indices visuels.

En ce qui concerne l'analyse spatiale, les indices visuels les plus fréquemment identifiés sur les objets dans une image sont les suivants :

- *Texture*. La texture des objets sur l'image est caractérisée par des variations locales de luminance. La texture est notamment utilisée depuis le traitement d'images médicales à un niveau microscopique (recherche de lésions au niveau cellulaire) jusqu'à l'analyse d'images satellites (nature des sols en télédétection) en passant par les images acquises depuis des caméras embarquées ou fixes [57]. Les indices de texture sont extraits couramment soit par les matrices de co-occurrence et soit par la technique de sommes et différences d'histogrammes.
- *Couleur*. La couleur est possiblement l'indice visuel le plus facile à exploiter dans un image. Il est normalement utilisé pour la détection et l'identification d'objets par des classifieurs paramétriques, de type bayésien, SVM ou AdaBoost. Quelques espaces de couleur utilisées dans la communauté de vision sont le RGB, le rgb normalisé, le HIS pour Teinte/Intensité/Saturation, le CIE-Lab, le CIE- Luv, l'espace de Ohta $I_1I_2I_3$... Un résumé des caractéristiques de chacun de ces espaces couleur est présenté dans [112].
- *Symétrie horizontale*. Cet indice peut être utilisé si la géométrie des objets structurés qu'on recherche dans l'environnement est symétrique et connue par avance. Par exemple, dans les travaux de Bertozzi et al. [17], les auteurs exploitent le fait que l'image d'un véhicule observée dans une vue frontale, présente une symétrie suivant un axe vertical. Cette symétrie est généralement estimée à partir des contours extraits des images.

Les approches spatiales utilisent avec plus de succès, la fusion de plusieurs caractéristiques décrites ci dessus notamment la texture et la couleur.

Nous avons testé et adapté dans nos propres travaux, une approche de classification couleur/texture ; cette méthode peut être considérée comme éparse [112], dans la mesure où elle s'appuie sur une segmentation préalable de l'image en régions uniformes en couleur. Dans ce travail l'image brute acquise par la caméra est d'abord acquise à partir d'un capteur mono-CCD, en utilisant un algorithme d'interpolation qui donne une image dans l'espace de couleur RGB. L'image RGB doit être bien distribuée en couleur car la technique de segmentation dépend directement de l'illumination et de la qualité de la couleur. Une image qui requiert une correction chromatique est présentée dans la figure 2.4 [2]. L'image 'a gauche est en format RGB mais il est notable qu'une dominante jaune prédomine partout dans l'image. L'image à droite montre cette même image après correction ; bien que cette image soit moins illuminée, elle a une meilleure distribution de la couleur, ce qui est requis pour obtenir de bonnes performances dans la segmentation.



(a) Image RGB (b) Image corrigé

FIGURE 2.4 – La correction d'une image pour une meilleure distribution de couleur. a) L'image RGB acquise par la caméra, b) L'image après correction.

Ensuite, l'image couleur corrigée est transformée vers l'espace $I_1I_2I_3$ proposé par OHTA [116] ; ce n'est pas l'espace optimal, mais c'est une transformation linéaire très rapide à appliquer, par rapport en particulier à la transformation vers CIE- Lab. Une méthode de segmentation trouve les régions sur l'image $I_1I_2I_3$ qui ont les attributs couleur les plus uniformes en fonction de seuils établis a priori. L'image des régions résultant de la segmentation de l'image 2.4b est montrée sur l'image en haut à gauche de la figure 2.5 et à la même position dans la figure 2.6. Notons que le nombre des régions trouvées dans l'image est très grand (près de 50 régions). Les couleurs utilisées pour distinguer ces régions, n'ont aucune signification ; elles servent uniquement pour distinguer les régions voisines entre elles. À cet étape, il devient nécessaire d'introduire un autre indice visuel qui permette d'identifier ces régions, puis de fusionner les petites régions trouvées sur une zone commune de l'image, par exemple sur le sol. Donc, sur chaque région segmentée dans l'image, les attributs de texture sont calculés pour tous les points de la région, et des attributs sont calculés pour la région. Ces attributs de texture sont concaténés avec les attributs de couleur déjà calculés dans l'étape de segmentation, afin de représenter les propriétés de chaque région.

Les vecteurs caractéristiques des régions sont utilisés d'abord dans une phase d'apprentissage pour construire une base de données sur les attributs de régions identifiées par un opérateur comme SOL, CIEL, ARBRE, HERBE, etc. puis dans une phase d'identification, par un classifieur fondé sur l'algo-

2. Cette séquence appartient à Thales Optolectronics ; elle a été acquise dans la cadre du projet TAROT

rithme de KNN (K-Nearest Neighbors) pour reconnaître la nature des objets et du terrain dans l'environnement d'un robot. Les résultats de la classification sont illustrés dans l'image à droite de la figure 2.5. La même image de régions a été testée sur un classifieur fondé sur la technique des SVM (Support Vectors Machine) ; les résultats correspondants sont illustrés en figure 2.6.

Tant la technique KNN que SVM sont des classifieurs supervisés qui exploite une base d'apprentissage pour générer une stratégie de classification ; dans tous les cas, la qualité de cette stratégie dépend des propriétés de la base (complétude, redondance, etc.). La méthode SVM trouve un classifieur optimal. Dans les figures 2.5 et 2.6, les régions identifiées de chaque classe sont distinguées par des couleurs différentes ; les régions rouges n'ont pas pu être classifiées. Après le codage des couleurs est le suivant : bleu pour le ciel, gris pour le sol et marron pour les arbres ou le bois, et vert pour la végétation.

En observant les résultats sur les images classifiées en figures 2.5 et 2.6, nous trouvons de notables erreurs de classification sur l'image traitée par les SVM, bien que les deux techniques utilisent la même base d'apprentissage. Cela montre la précaution qu'on doit prendre au moment de l'apprentissage ; les échantillons doivent être mieux sélectionnés pour le cas de la méthode SVM. Finalement, les deux figures montrent dans l'image du milieu la région identifiée comme le sol de la scène ; la technique KNN permet d'identifier mieux le sol que les SVM.

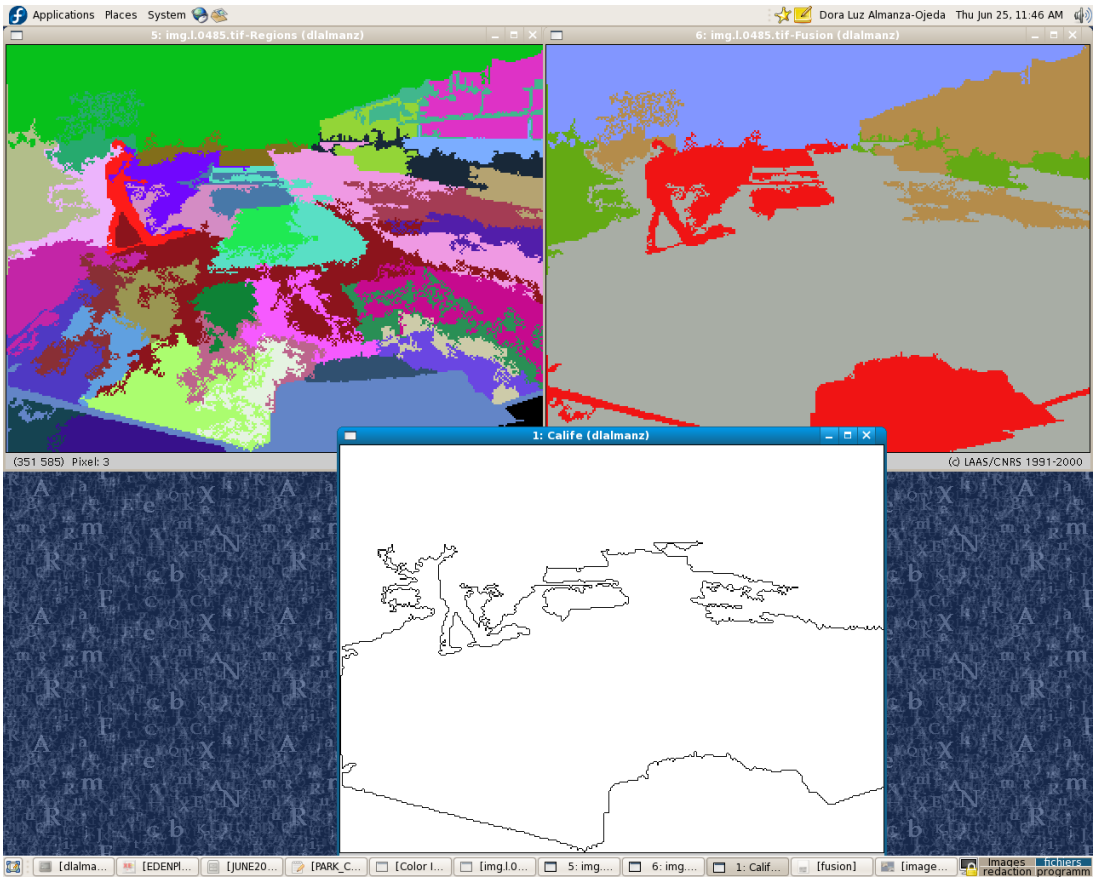


FIGURE 2.5 – Images résultant de la classification par KNN en utilisant l'information couleur-texture de l'image de la figure 2.4b. En haut à gauche l'image de segmentation par couleur uniquement, à droite cette même image classifiée par KNN, les régions en rouge représentent les zones non classifiées. En bas au milieu les contours de la région classée SOL, donc libre d'obstacle.

Ls approches temporelles exploitent le mouvement apparent dans une séquence d'images, comme

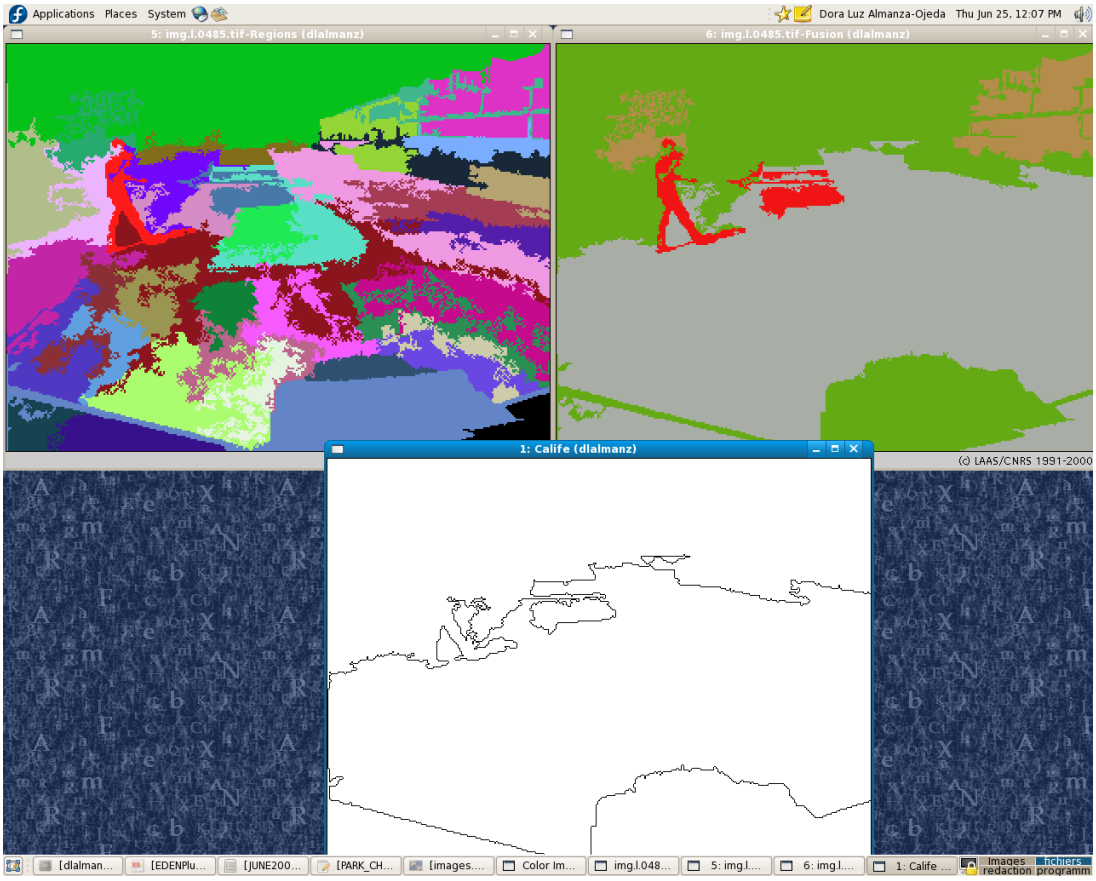


FIGURE 2.6 – Images résultant de la classification par SVM en utilisant l'information couleur-texture de l'image de la figure 2.4b. En haut à gauche l'image de segmentation par couleur uniquement, à droite cette même image classifiée par SVM. En bas au milieu les contours de la région classée SOL.

caractéristique d'intérêt. Dans le cas où la caméra est embarquée sur un véhicule ou sur un robot, nous pouvons rencontrer certaines difficultés à distinguer le mouvement propre du véhicule de celui des éléments la scène, puis à estimer à partir de l'image, le mouvement d'autres objets mobiles, puisqu'une image donne juste une projection de la scène 3D. Les effets perspective introduits par la projection, engendrent des mouvements apparents différents selon l'endroit où se trouvent les objets détectés dans l'image. C'est ce qu'on appelle l'effet de parallaxe : pour un même mouvement 3D, les objets proches de la caméra engendreront un mouvement apparent dans l'image plus important que celui des objets lointains. Par ailleurs, une hypothèse de mouvement linéaire (ou mouvement à vitesse constante) est normalement utilisée avec pour but d'estimer le déplacement apparent v dans le temps des indices visuels. Ce déplacement est obtenu en calculant la différence entre la position de cet indice au temps t et sa position au temps $t+1$; cela implique que cet indice a pu être apparié ou suivi entre ces images successives de la séquence.

Les meilleures performances pour la détection d'obstacles par vision monoculaire, sont obtenues par des approches hétérogènes fondées sur l'analyse spatiale et temporelle. Nous évoquons, pour ce type d'approche, le travail développé dans [133] où le vecteur v est obtenu par une minimisation aux moindres carrés sur un groupe de pixels décrivant l'indice visuel (une fenêtre gaussienne) entre les images à deux temps successives. Une autre approche spatio- temporelle, que d'ailleurs nous avons analysée et mise en place dans nos travaux (décrits dans le chapitre 3), a été proposée par Veit et al. [140] ; jes auteurs utilisent

trois descripteurs depuis lesquels ils extraient leurs modèles de mouvement apparent dans une séquence d'images. Ces mouvements sont comparées avec un modèle aléatoire qui représente le mouvement apparent pour les objets statiques appartenant au fond ou arrière-plan de l'image. Cette comparaison permet au système de distinguer entre les mouvements d'objets du fond (y compris le va-et-vient des feuilles d'un arbre, les petites changements d'illumination dans la scène, etc.) qui n'ont pas d'importance dans la détection d'objets mobiles, et les mouvements d'un obstacle dynamique potentiel.

2.3.4 Conclusion pour les capteurs visuels.

Nous avons décrit les différentes méthodes proposées en vision dans le contexte de la robotique mobile, notamment pour la détection d'objets et la classification des régions segmentées sur les images. D'une part les inconvénients des bancs stéréo sont devenus de moins en moins gênants avec les nouveaux dispositifs commerciaux pré calibrés : donc ils sont des candidats très pertinents pour les applications robotiques, bien que cela reste encore un système pas très acceptable économiquement, notamment sur des véhicules grand public. D'autre part utiliser une seule caméra devient un challenge pour les applications d'extérieur du fait des conditions d'illumination et d'une excessive variation des caractéristiques de texture.

Dans notre travail, nous allons exploiter un système multi-caméras, mais qui n'est pas un système stéréo du fait du non recouvrement des champs de vue des caméras. En recourant à des architectures dédiées exploitant des technologies programmables comme les FPGA, nous proposons de grandes améliorations sur le temps de traitement es images, ce qui rend possible de traiter en parallèle un grand nombre d'images. Cependant le coût et le temps de la développement d'une solution sur de telles architectures demeurent plus élevés que pour le développement d'un logiciel.

Nous décrivons dans la section suivante les approches spatio-temporelles utilisées dans notre système intégré multi-caméras pour la détection et le suivi d'obstacles depuis un robot de service. Nous verrons dans les chapitres suivants comment adapter ces approches pour les développer à moindres coûts sur des architectures dédiées.

Chapitre 3

Algorithmes pour la détection d'obstacles

Sommaire

3.1 Introduction

Ce chapitre a pour but de présenter brièvement dans un premier temps l'algorithme global pour la détection d'obstacles, lequel est composé de deux méthodes indépendantes exécutées sur chaque image acquise par les N micro-caméras situées autour du robot mobile. Puis dans un deuxième temps et de manière plus détaillée, décrire les différents algorithmes qui intègrent les deux méthodes utilisées pour la détection. Enfin, présenter les résultats expérimentaux obtenus à partir des différents algorithmes qui intègrent le système de détection d'obstacles.

L'algorithme global (voir figure 3.1) utilisé pour la détection d'obstacles est composé de N processus parallèles et indépendants, chacun d'eux correspondant à la détection d'obstacles pour chacune des N micro-caméras situées autour du robot mobile. Le nombre N de micro-caméras dépend du design du robot : il faut autant de micro-caméras que le nécessite la surface à couvrir. La figure 3.2 représente la simulation d'un exemple de ceinture de micro caméras utilisé pour la détection d'obstacles depuis un robot circulaire : le parallélépipède bleu représente le robot et les micro-caméras autour du robot sont représentées par plusieurs parallélépipèdes rouges. Celles-ci sont nécessaires pour couvrir la surface

autour du robot (en jaune sur la figure). Les angles morts qui existent entre les champs de vue dans ce cas, peuvent être éliminés soit en augmentant le nombre de caméras et en modifiant leur positionnement géométrique soit en changeant les angles d'ouverture des optiques. Tous les aspects de la sélection des micro-caméras et la conception du capteur seront traités plus en profondeur dans le chapitre 5. Nous ne considèrerons, pour l'instant, qu'une notion de N flux parallèles et indépendants issus de N micro-caméras réparties autour du robot.

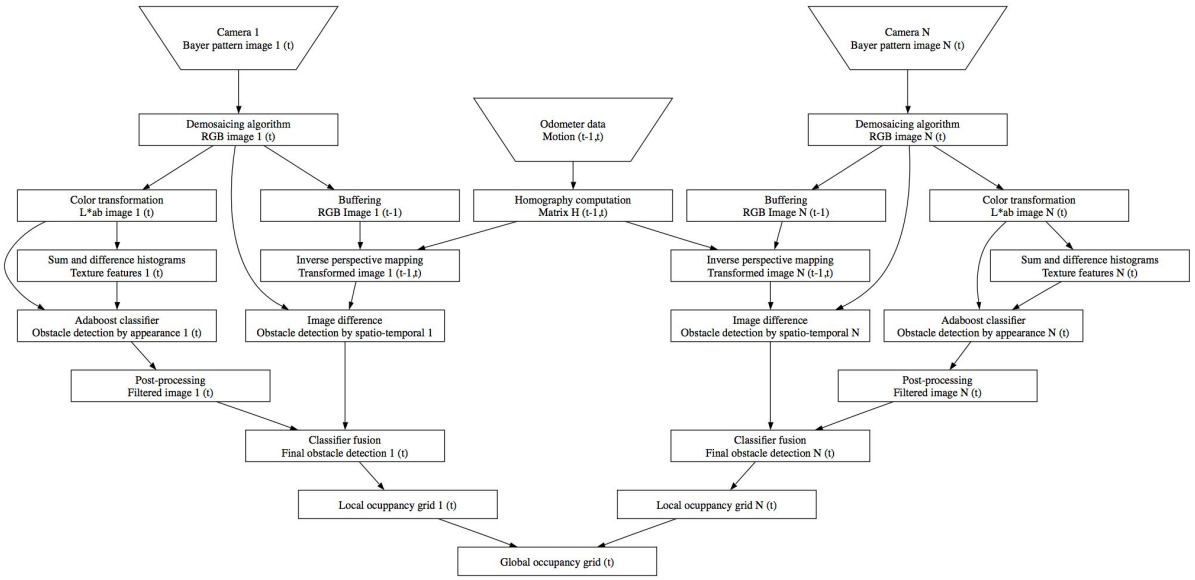
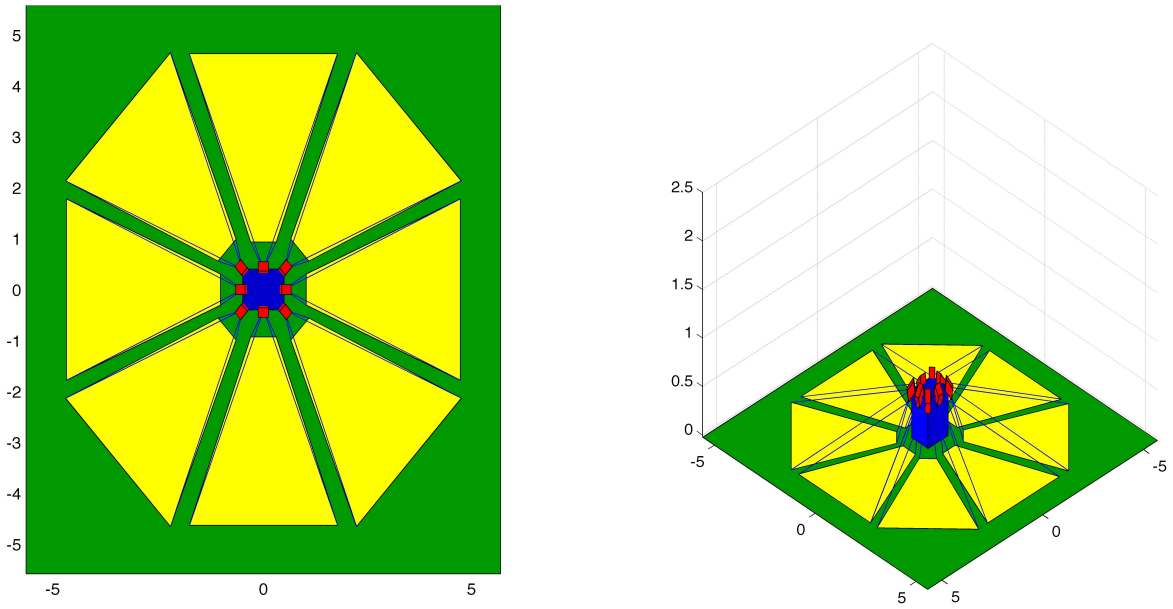


FIGURE 3.1 – Diagramme global de l'algorithme pour la détection d'obstacles.

Tout d'abord, les images acquises sont traitées numériquement pour les rendre exploitables par les algorithmes de détection d'obstacles. L'image couleur est reconstruite à partir d'une image monochromatique brute (filtre *Bayer*) en utilisant une méthode d'interpolation appelée demosaïquage. Pour chacune des images couleur acquises par les micro-caméras), deux méthodes de détection indépendantes sont exécutées en parallèle.

La première méthode utilise les caractéristiques de la couleur et de la texture de l'image. Cette méthode est basée sur l'hypothèse que le modèle de couleur et de texture du sol est connu, ce qui rend possible l'utilisation d'une technique d'apprentissage automatique et ainsi faire la discrimination des obstacles et du sol. Il convient de rappeler que pour obtenir une bonne performance avec cette technique, il faut avoir un grand nombre d'échantillons pour les différents obstacles et le sol. Dans cette méthode, on peut assimiler les obstacles à n'importe quel objet qui n'a pas la couleur et la texture du sol. Par ailleurs nous pourrions traiter de l'identification des obstacles à partir de leur apparence, ce qui améliorerait la robustesse par rapport à une simple détection. Cependant cette méthode a un défaut important : elle peut détecter des faux obstacles dans le cas d'ombres ou de taches présentes sur le sol : Une deuxième méthode doit donc être mise en œuvre pour corriger ces possibles erreurs de classification.

Cette deuxième méthode utilise une analyse spatio-temporelle. Elle est basée sur le mouvement du robot. L'image couleur acquise depuis une micro-caméra dans l'instant t est projetée sur le sol puis comparée avec l'image couleur acquise par la même micro-camera dans l'instant $t-1$. Cette dernière image couleur est projetée deux fois : la première vers le sol et la deuxième vers le temps t. La dernière projection utilise l'information de l'odomètre entre les temps t et $t-1$. En supposant qu'aucun changement dans l'illumination de la scène n'a eu lieu entre les instants d'acquisition $t-1$ et t, alors la différence entre les pixels qui sont au niveau du sol dans l'image acquise au temps $t-1$ et les pixels correspondants



(a) (b)

FIGURE 3.2 – Simulation de la zone couverte par la ceinture de micro-caméras placée autour du robot, (a) vue d'en haut, (b) vue à partir de $45°$

dans l'image acquise au temps t sera proche de zéro. Par contre les pixels situés au-dessus ou au-dessous du niveau du sol seront différents de zéro et les objets qui leur sont associés seront considérés comme des obstacles réels.

Les deux méthodes présentées sont fusionnées pour obtenir une seule image de détection pour chacune des caméras ; chaque pixel code la probabilité qu'il corresponde à un point du sol. Le résultat de la fusion a les avantages des deux méthodes, ce qui permet d'obtenir une détection plus robuste en réduisant les erreurs de classification. L'étape suivante est la projection vers le sol de l'image de détection trouvée pour chacune des caméras, dans le but de construire une grille d'occupation ; cette grille peut être construite dans le repère de l'environnement, ou peut être robot-centrée.

La dernière étape est la fusion et la construction de la grille d'occupation globale à partir des N grilles d'occupation locales résultant de la fusion des deux méthodes de détection d'obstacles pour chacune des N micro caméras situées autour du robot. La grille d'occupation globale à l'instant t est calculée en fonction de la grille d'occupation globale de l'instant $t-1$ recalée si elle est robot-centrée sur la position courante du robot, et de chacune des grilles locales construites pour chacune des caméras à l'instant t.

Dans les sections suivantes, nous décrirons et comparerons tout d'abord les méthodes que nous avons utilisées pour reconstituer une image couleur à partir d'une mosaïque *Bayer*. Puis nous traiterons du calcul des attributs de couleur et texture utilisés dans la détection d'obstacles par l'apparence. Ensuite nous décrirons l'approche géométrique. Enfin, nous présenterons les résultats expérimentaux des deux méthodes et la grille d'occupation locale qui en résulte.

3.2 Détection d'obstacles pour une approche basée sur l'apparence

La texture et la couleur sont deux des plus importants attributs utilisés en traitement des images et en reconnaissance des formes pour faire de la détection et de l'identification d'objets dans une scène. Notre approche utilise aussi ces attributs pour détecter les pixels correspondant à des points du sol,

sachant qu'on connaît à l'avance un modèle approché du sol. En utilisant une technique d'apprentissage automatique il est possible de discriminer les différents objets présents dans une scène et donc d'effectuer la classification de ces objets comme des obstacles ou non.

Mais comment calculer ces attributs ? Dans la communauté de vision par ordinateur, la plupart des approches utilisent au début une technique de segmentation, le plus souvent fondée sur la couleur. Cette technique partitionne l'image en régions selon un critère d'uniformité chromatique. Cette étape permet de réduire le temps de calcul nécessaire ensuite pour la classification, qui est appliquée uniquement sur les régions et non sur chaque pixel [10, 9]. Toutefois, dans certains cas, cette approche fonctionne mal, surtout dans le cas de la robotique d'intérieur où le sol peut être très texturé et le contraste des couleurs sur le sol très marqué. Dans ce cas la segmentation utilisant la couleur donnerait beaucoup de petites régions, ce qui reviendrait à augmenter le temps de calcul.

Une deuxième technique consiste à diviser l'image dans des voisinages carrés, sans entrelacement, dans le but de réduire le nombre de pixels à classer et donc le temps de calcul. Cette technique est basée sur le principe que, dans le voisinage d'un pixel, il n'y a que des pixels appartenant à la même classe et il est donc possible de calculer des attributs moyens sur un voisinage et de classifier sur la base de ces attributs [5]. La technique dépend de la manière de définir les voisinages ; elle nécessite de trouver un bon compromis, car la performance en temps de calcul est directement proportionnelle à la probabilité d'erreur dans la classification, c'est à dire, plus le voisinage est grand, plus la probabilité de se tromper est importante, et meilleurs sont les temps de calcul.

Une troisième technique, moins utilisée, est la classification dense (c'est-à-dire pixel par pixel) ; les attributs sont calculés dans des voisinages centrés sur chaque pixel. Elle demande un grand nombre d'opérations et donc un important temps de calcul sur un ordinateur conventionnel. C'est la raison pour laquelle elle est peu exploitée pour des applications telles que la navigation de véhicules autonomes, la surveillance vidéo ou l'imagerie médicale, qui toutes, nécessitent des performances en temps réel que les ordinateurs conventionnels ne peuvent pas garantir. Néanmoins la classification dense a l'avantage d'être plus structurée et donc plus simple à mettre en œuvre sur une plateforme de traitement intégré et parallèle.

Les algorithmes développés et mis en œuvre dans nos travaux pour la détection d'obstacles, sont basés sur une approche dense afin d'obtenir une bonne robustesse de détection. Ces algorithmes seront exécutés sur des architectures matérielles dans le but de satisfaire les contraintes temps réel liées à la robotique mobile.

3.2.1 Démosaïquage : reproduction des images couleur

L'acquisition d'une image couleur peut se faire à l'aide de plusieurs types de caméras. Nous n'avons exploité que des caméras CMOS (Complementary metal oxide semi-conductor) numériques avec liaison parallèle [130, 134]. Les capteurs CMOS sont apparus dans les années 1980, à la suite des matrices de photodiodes comme le résultat de l'intégration de cellules composées d'une photodiode et d'une logique d'amplification puis d'obturation. Ils sont plus complexes à fabriquer mais sont produits selon des techniques classiques de micro-électronique et de ce fait peuvent avoir des dimensions importantes (24 méga pixels). La méthode la plus répandue pour obtenir des images couleur avec un capteur CMOS est la même que pour beaucoup de CCD : elle consiste à placer devant chaque cellule sensible, un filtre du type CFA de telle sorte que chaque photodiode du capteur CMOS ne perçoit qu'une des trois composantes spectrales, généralement Rouge, Verte et Bleu. Les pixels sont alors disposés selon un matriçage dit en quinconce.

Pour la mosaïque d'une matrice colorée, le filtre le plus utilisé est le 3-chromatique RGB, bien que autres soient aussi disponibles : le 3-couleurs complémentaires YeMaCy, le système à 4-couleurs où la quatrième couleur est le blanc ou une autre couleur d'une sensibilité spectrale décalée [4]. Bien que

l'utilisation de plus de 3 composantes colorimétriques, dans la fabrication de capteurs, semble donner plus d'informations spectrales sur la scène, la corrélation implicite entre les composantes de couleur réduit son utilisation en pratique.

Les algorithmes d'interpolation des matrices couleur transforment la sortie du capteur mono-CMOS en une vraie image couleur (i.e., RGB codé en 24 bits par pixel), en reconstruisant les trois composantes chromatiques sur chaque pixel (voir figure 3.3).

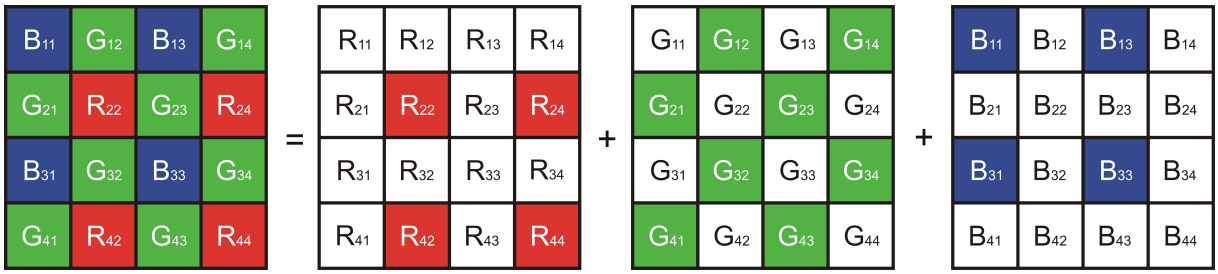


FIGURE 3.3 – Démosaïquage : recuperation de l'information couleur manquente.

Plusieurs approches sont inspirées du système visuel humain. En effet la rétine de notre oeil, qui contient les récepteurs visuels, peut être comparée à une matrice de filtres couleurs, car chacun des photorécepteurs est sensible à une seule gamme de longueur d'onde [120]. Il a été montré par les expériences de psychologie que ces photorécepteurs peuvent avoir une grande acuité spatiale en luminance et en couleur selon les techniques de démosaïquage [4].

La relation d'interpolation peut être exprimée par l'équation 3.1, où $c\left(u,v\right)$ représente le pixel interpolé, $c\left(u_k,v_l\right)$ est la valeur échantillonnée sur le pixel $\left(u_k,v_l\right)$ et h représente le noyau d'interpolation appliqué sur une fenêtre $M \times N$ autour du pixel interpolé [123].

$$c\left(u,v\right) = \sum_{k=1}^{M}\sum_{l=1}^{N} c\left(u_k,v_l\right) h\left(\left|u-u_k\right|\right) h\left(\left|v-v_k\right|\right) \tag{3.1}$$

Nous avons étudié le comportement de trois algorithmes d'interpolation en fonction de critères comme la complexité algorithmique, le temps d'exécution sur une plateforme dédiée, la qualité de la reconstruction visuelle et l'impact sur la classification. Notre objectif est de sélectionner la méthode la plus adaptée pour notre problème de détection d'obstacles.

Parmi les méthodes testées, nous ne détaillerons ici que les suivantes : une interpolation simple par le plus proche voisin, une interpolation linéaire par un filtrage type moyenne et une interpolation bilinéaire.

Démosaïquage par le plus proche voisin (PPV)

Dans cet algorithme, la valeur des composantes rouge (R), verte (G) ou bleu (B) pour le pixel interpolé, se calcule à partir de la valeur de l'un des pixels (de la même classe) le plus proche dans son voisinage, c'est-à-dire que leur implémentation implique une réplication des valeurs. Il est alors possible de choisir ce voisin, quelle que soit sa position : supérieure, inférieure, gauche ou droite. Le noyau d'interpolation [123] pour cette méthode est régi par l'équation suivante :

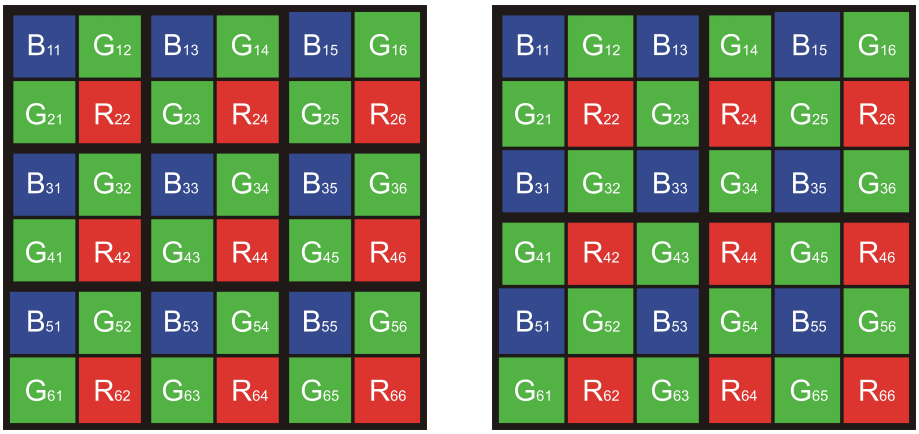
$$h\left(x\right) = \begin{cases} 0 & 0 \leq x \leq 0,5 \\ 1 & x > 0,5 \end{cases} \tag{3.2}$$

Grâce à cette opération, appliquée sur toutes les photodiodes de la matrice CMOS, nous obtenons les trois couleurs de la scène. En dépit de sa simplicité algorithmique, cette méthode a l'inconvénient de produire des images de qualité médiocre. Elle est donc déconseillée pour le traitement d'images.

Démosaïquage linéaire

Dans cette approche linéaire, la valeur des composantes rouge ou bleu pour le pixel interpolé, se calcule de la même façon que dans l'approche PPV, sauf que les valeurs ne sont pas répliquées, elles sont calculées dans un voisinage de 2x2 pixels. La composante verte est obtenue par moyennage des deux pixels (de la même classe de photorécepteurs) pris dans son voisinage. Le noyau de cette interpolation est présenté graphiquement dans la figure 3.4a. Les composantes rouge, verte et bleu sont obtenues par les équations 3.3.

$$h_R(x) = \left\{ \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad h_G(x) = \left\{ \frac{1}{2} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad h_B(x) = \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}. \right. \right. \right. \tag{3.3}$$



(a) (b)

FIGURE 3.4 – Mosaïque Bayer BGGR, Démosaïquage (a) linéaire, (b) bilinéaire

La complexité algorithmique de cette méthode est plus grande que la méthode PPV et de plus la méthode réduit la résolution de l'image bayer. Cependant, la prise en compte d'un voisinage plus grand rend la méthode moins sensible au bruit qu'avec PPV.

Démosaïquage bilinéaire

Cette approche linéaire [98] utilise l'information des quatre pixels adjacents (de la même classe de photorécepteurs) pour calculer le pixel manquant par moyennage. Le noyau de cette interpolation est formulé [123] dans l'équation 3.4 et présenté graphiquement dans la figure 3.4b,

$$h(x) = \begin{cases} 1 - x & 0 \leq x \leq 1, \\ 0 & 1 < x. \end{cases} \tag{3.4}$$

La composante verte est explicitement estimée par l'équation 3.6 sur les pixels R ou B, alors que les composantes rouge et bleue sont obtenues par les équations 3.5 et 3.7, selon la nature du pixel considéré.

$$h_R(x) = \begin{cases} \frac{1}{2} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} & x \in x_G, \\ \frac{1}{4} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} & x \in x_B. \end{cases} \tag{3.5}$$

$$h_G(x) = \begin{cases} \frac{1}{4}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} & x \in \{x_R, x_B\}. \end{cases} \tag{3.6}$$

$$h_B(x) = \begin{cases} \frac{1}{2}\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} & x \in x_G, \\ \frac{1}{4}\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} & x \in x_R. \end{cases} \tag{3.7}$$

Notons que ce processus est proche d'un filtre passe bas avec une bande passante limitée, ce qui génère un lissage colorimétrique des frontières en produisant des fausses couleurs surtout sur les images texturées. Néanmoins, dans le cas du sous échantillonnage la prise en compte des quatre voisins rend la méthode moins sensible au bruit qu'avec les approches évoquées dans les sections précédentes.

3.2.2 Calcul des attribut de couler

La couleur est un attribut visuel perçu par les humains comme une combinaison tri-stimuli (R, G, B) : ces trois composantes sont appelées les couleurs primaires. Les composantes tri-stimuli (R, G, B) sont représentées par les valeurs de luminosité dans la scène. Elles sont obtenues en utilisant trois filtres séparés et centrés sur différentes longueurs d'onde λ (425,8 nm pour le bleu, 546,1 nm pour le vert et 650,0 nm pour le rouge),

$$R = \int_\lambda E(\lambda) S_R(\lambda) d\lambda \quad G = \int_\lambda E(\lambda) S_G(\lambda) d\lambda \quad B = \int_\lambda E(\lambda) S_B(\lambda) d\lambda \tag{3.8}$$

où S_R, S_G at S_B sont les filtres colorés placés devant le spectre de la lumière incidente. Pour des opérations d'affichage, cet espace est très utilisé (écrans d'ordinateur, systèmes de télévision, appareils photo et cameras numériques) ; en revanche, la corrélation entre ses composantes chromatiques le rend inadéquat lors des changements d'intensité liés à la classification ou segmentation couleur. Une corrélation implique une interdépendance entre les changements d'intensité et les variations de chacune des composantes chromatiques ; la représentation RGB est donc très sensible aux changements d'illumination [19]. Ainsi, la distance entre couleurs dans cet espace ne représente pas une mesure perceptiblement uniforme des différences de couleur.

Il existe de nombreuses autres représentations de la couleur utilisées en traitement d'images (HSI, $I_1I_2I_3$, $CIE{-}L^*u^*v^*$, $CIE{-}L^*a^*b^*$, ...) ; il n'existe pas de consensus dans la communauté scientifique, sur la *meilleur* représentation qui se révélerait supérieure en performance, surtout pour coder la couleur dans des images acquises sur des scènes réelles. La sélection de l'espace de couleur est donc le premier choix important à prendre en compte lors de la mise en œuvre d'une technique de classification ou de segmentation.

Les représentations de la couleur sont classées en deux groupes : les espaces de couleur linéaires et non-linéaires.

Les principales représentations linéaires de la couleur sont :

- Le système *XYZ*, dites coordonnées primaires. La composante Y représente approximativement la sensibilité de l'œil humain à la luminosité (considérée comme la composante de luminance du spectre incident).
- Le système *YIQ* est utilisé pour codifier l'information de couleur sur les signaux des systèmes américains de télévision (NTSC). La composante Y représente la luminance tandis que I et Q

sont respectivement les composantes chromatiques représentant les oppositions cyan-orange et magenta-bleu.
- L'espace *YUV* est le standard adopté pour les systèmes européens de télévision (PAL). Les composantes *YUV* ont la même signification que le standard *YIQ* mais la particularité de cet espace consiste à utiliser le blanc comme référence D_{65}.
- La représentation *YES* a été conçue par la SMPTE (Society of Motion Picture and Television Engineers) et utilisée par XEROX. Elle contient la luminance Y, les composantes de chrominance E (l'axe rouge-vert) et S (l'axe jaune-bleu).
- L'espace $I_1I_2I_3$ développé par Ohta [116] est le résultat d'une expérimentation avec une certaine quantité d'attributs couleur qui ont été utilisés pour évaluer une méthode de segmentation sur différents types d'images. Cet espace provient d'une transformation de Karhunen-Loéve pour déterminer les attributs couleur non corrélés (orthogonaux) et les plus discriminants.

Les transformations non-linéaires de couleur sont plus souvent utilisées pour les applications de traitement d'images que les transformations linéaires. Ceci s'explique par plusieurs raisons : elles sont parfois plus robustes aux changements d'illumination et plusieurs d'entre elles se sont inspirées de la perception visuelle de la couleur chez l'être humain. Les espaces de couleur non-linéaires les plus utilisés sont :

- Les applications réelles ont besoin d'un espace robuste aux changements d'éclairage. L'espace normalisé *rgb* a été conçu pour obtenir des variations d'intensité uniformes sur toute la distribution spectrale de couleur.
- La représentation $l_1l_2l_3$ proposée par T. Gevers [50] a trouvé des applications dans la reconnaissance d'objets grâce à une certaine robustesse aux changements d'illuminations.
- L'espace de couleur *CIE-Lab* fut développé pour représenter une uniformité perceptuelle de la couleur compatible avec la notion psychophysique de la couleur chez l'humain.
- L'espace de couleur *CIE-L*uv* est défini de façon similaire à *CIE-Lab*, en particulier le calcul de la luminance $L*$ utilise la même expression. Ces deux derniers espaces sont particulièrement efficaces pour mesurer de petites variations de la couleur mais les problèmes de singularité sont encore un des points faibles de ces approximations.
- L'espace *ITS* est un espace qui reste très utilisé dans des applications de traitement d'images. Il correspond davantage à la notion habituelle de couleur chez l'homme, où les variations de saturation sont facilement détectées. Malheureusement il n'existe pas une formulation mathématique unique et plusieurs variations peuvent donc être disponibles.

Notre méthode de détection a besoin d'un espace de couleur le plus indépendant possible du changement d'illumination. De plus cette transformation de la couleur doit être simple à mettre en œuvre sur une plateforme embarquée. Pour ces raisons, nous avons choisi l'espace de couleur *CIE-Lab*, qui donne le meilleur rapport d'immunité au changement d'éclairage et de simplicité pour l'implémentation vers une plateforme embarquée. Une description plus détaillée de l'espace de couleur *CIE-Lab* sera présenté dans la section suivante.

Espace de couleur CIE-Lab

L'espace de couleur *CIE-Lab* (plus précisément *CIE-L*a*b**) fut développé en 1976 par la Commission Internationale de l'Eclairage (CIE) [71]. Comme tout les systèmes issus du système *CIE-XYZ*, il caractérise une couleur à l'aide d'un paramètre d'intensité correspondant à la luminance et de deux paramètres de chrominance qui décrivent la couleur. Il a été spécialement étudié parce que les distances calculées entre couleurs correspondent aux différences perçues par l'œil humain. Il essaye de prendre en compte la réponse logarithmique de l'œil ; de ce fait, il est très utilisé dans le cas de mélanges de pigments (industrie graphique, photographie, restauration de peintures, autres).

Le modèle *CIE-Lab* s'appuie sur la théorie des opposants en s'inspirant de l'ancien modèle *Lab* (1942) de Richard Hunter. Afin de ne pas le confondre avec son prédécesseur, il est noté avec des astérisques soit *L*a*b**. Mais aujourd'hui, il est simplement appelé *CIE-Lab* ou *CIE-L*ab*, le nom officiel étant *1976 CIE-L*a*b**. C'est dans les années 60 qu'on s'est rendu compte que la théorie des opposants correspondait à la réception des couleurs au niveau du cerveau qui oppose le noir au blanc, le bleu au jaune et le rouge au vert. Le *CIE-Lab* s'appuie sur cette théorie et se distingue de ce fait du modèle *CIE-XYZ*.

Nous utilisons cet espace pour corriger en ligne les déséquilibres chromatiques, provoqués en particulier par les variations d'illumination. Afin de définir la transformation de couleur, la première étape nécessaire est la sélection d'un illuminant standard. Un illuminant standard est une source théorique de la lumière visible avec profil (distribution spectrale de puissance) qui est publié. Les illuminants standards fournissent une base pour comparer les images ou les couleurs enregistrées sous un éclairage différent. Nous avons choisi l'illuminant standard *CIE D65* (parfois écrit *D65*) qui est un illuminant standard couramment utilisé et défini par la CIE. Il fait partie de la série *D* des sources lumineuses qui tentent de représenter une illumination standard pour des conditions d'éclairage en plein air dans différentes parties du monde [115].

Le *D65* correspond en gros à un soleil de midi en Europe occidentale / Europe du Nord. Comme tout illuminant standard il est représenté sous la forme d'un tableau de données spectro-photométriques moyennes. Une source de lumière qui a statistiquement la même distribution spectrale relative de puissance (SPD), peut être considérée comme une source de lumière *D65*. Il n'existe aucune source réelle de lumière *D65*, mis à part des simulateurs. La qualité d'un simulateur peut être évaluée par l'indice CIE métamérisme. Le *D65* est destiné à représenter la lumière du jour avec une température de couleur d'environ 6500 K. L'illuminant standard *D65* doit être utilisé dans tous les calculs colorimétriques liés à un contexte de lumière du jour, à moins que des raisons particulières poussent à utiliser un illuminant différent [115].

La transformation de couleur *CIE-Lab* est une fonction des coordonnées primaires *XYZ* avec l'illuminant standard *D65*. Les coordonnées primaires *XYZ* sont calculées par une transformation linéaire à partir des composants tri-stimuli *RGB* (couleurs normalisées). Elle s'obtient grâce à la matrice suivante :

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{3.9}$$

La luminosité *L** et les composantes chromatiques *a* et *b* sont données par les équations suivantes en fonction des coordonnées primaires *XYZ* :

$$L^* = 116 \left[f\left(\frac{Y}{Y_w}\right) \right] - 16 \tag{3.10}$$

$$a = 500 \left[f\left(\frac{X}{X_w} - f\left(\frac{Y}{Y_w}\right)\right) \right] \tag{3.11}$$

$$b = 200 \left[f\left(\frac{Y}{Y_w} - f\left(\frac{Z}{Z_w}\right)\right) \right] \tag{3.12}$$

où X_w, Y_w at Z_w symbolisent le blanc de référence qui dans l'illuminant standard *D65*, a été fixé à (0.950456, 1.000000, 1.088754). Notons que la fonction $f(t)$ va gérer la stabilité (ou le bruit) de cet espace à basse intensité ; elle est définie comme suit :

$$f(t) = \begin{cases} t^{\frac{1}{3}} & si\ t > \left(\frac{6}{29}\right)^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{4}{29} & si\ t \leq \left(\frac{6}{29}\right)^3 \end{cases} \tag{3.13}$$

La racine cubique utilisée dans ces équations est très intéressante car des expériences psycho-visuelles ont montré un comportement non linéaire analogue à celui de l'œil humain. De plus, l'introduction du rapport $\frac{Y}{Y_w}$ permet de simuler grossièrement l'adaptation de l'œil humain à une luminosité de référence.

La luminance L^* donne la notion de clarté, qui va de 0 (noir) à 100 (blanc), tandis que la chrominance est représentée par a et b qui sont respectivement l'opposition de couleur vert (valeur négative) - rouge (valeur positive) et l'opposition bleu(valeur négative) - jaune(valeur positive) en passant par le blanc (0) si la clarté vaut 100 (voir figure 3.5).

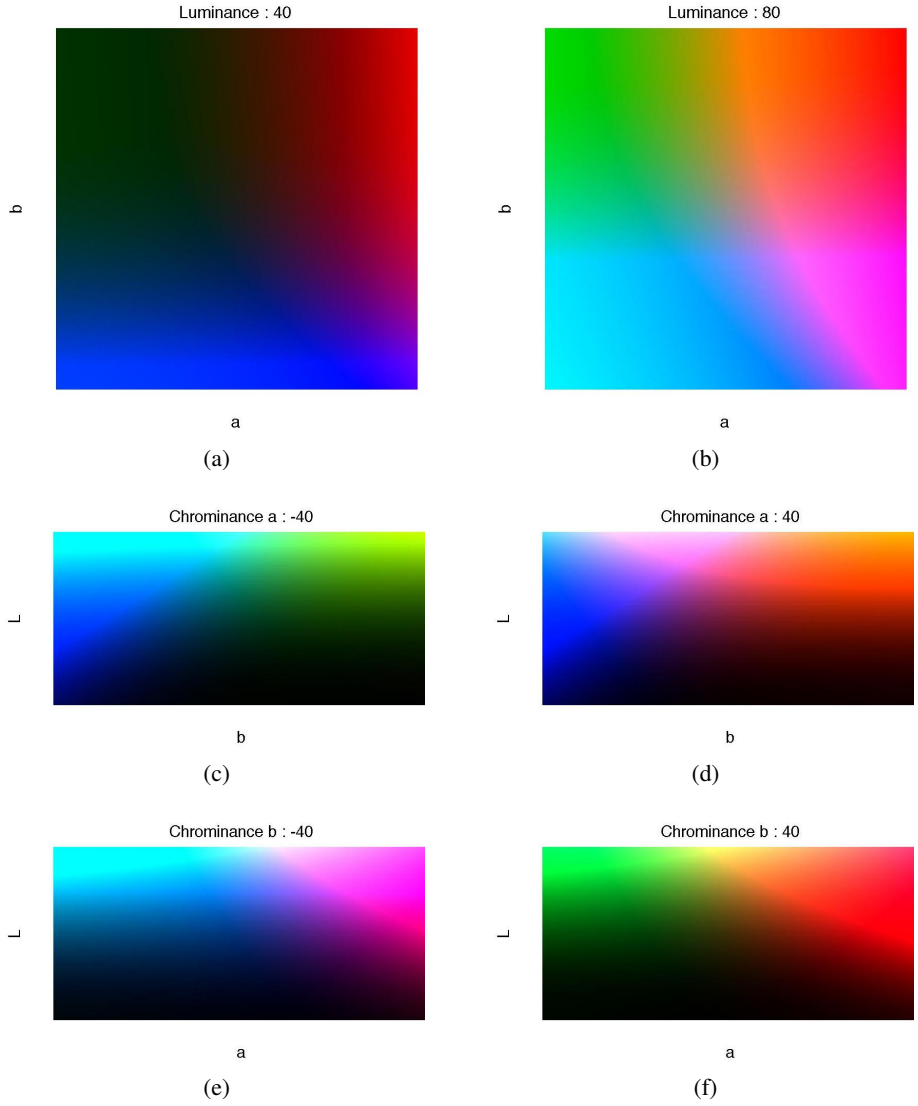


FIGURE 3.5 – Représentation de l'espace de couleur *CIE-Lab* pour différentes valeurs de luminance et chrominance pour l'illuminant *D65*.

La notion d'uniformité est mieux définie que dans l'espace *RGB* : la distance euclidienne entre deux couleurs, ou écart visuel, devrait être perçue par l'œil humain avec la grandeur définie comme suit,

$$\Delta E_{ab} = \sqrt{\Delta L^{*2} + \Delta a^2 + \Delta b^2} \tag{3.14}$$

Il est possible de définir la notion de teinte et de saturation à partir des composantes a et b. La teinte est définie par la valeur angulaire :

$$h = \tan^{-1}\left(\frac{b}{a}\right) \tag{3.15}$$

et la saturation ou chroma (niveau de coloration) par la grandeur :

$$C = \sqrt{a^2 + b^2} \tag{3.16}$$

Les composantes chromatiques a et b sont utilisées durant le processus de classification comme caractéristiques de couleur et la luminance L^* est utilisée pour calculer les attributs de texture, qui seront expliqués dans la section suivante. Nous avons sélectionné la luminance L^* pour calculer les attributs de texture car elle présente la plus grande variation de contraste sur les régions texturées et donc la modélisation de textures est plus représentative sur cette composante.

3.2.3 Calcul des attributs de texture

Bien que la texture soit visible de manière naturelle par le système visuel humain, en donner une définition est très complexe. Cette difficulté est mise en évidence par les diverses définitions de texture disponible dans la littérature. Ainsi, actuellement, il n'existe aucune règle mathématique de caractère général qui soit capable de quantifier les différentes classes de texture. Cependant, les propriétés intuitives les plus utilisées pour décrire la texture sont les suivantes :

- La texture est une propriété liée à une région support, qui doit être uniforme au sens sémantique. Elle n'est donc pas définie en un point, mais sur un ensemble de pixels dans un voisinage spatial de ce point. Par exemple, calculer la texture sur une région d'image acquise pour une partie sur le sol uniforme, et pour une autre, sur n'importe quel objet, n'a pas de sens ; les régions support pour le calcul de la texture doivent donc être bien définies.
- La texture est le reflet de l'analyse d'une distribution spatiale des pixels en nivaux de gris dans la région support, c'est à dire, des histogrammes ou des matrices bidimensionnelles de co-occurence.
- La texture dans une image peut être perçue et mesurée à différents nivaux de résolution.
- Une région est dite texturée quand le nombre de primitives caractérisant l'objet dans la région est grand et quand aucune forme individuelle (significative) n'est présente.

De nombreux travaux ont été dédiés à ce sujet ce qui nous permet de disposer de différents opérateurs pour mesurer la texture :

1. **Approches structurelles géométriques.** Ces approches considèrent que les textures sont formées par des motifs ou des primitives réguliers qui se répètent périodiquement (ou presque) dans une région limitée de l'espace, par exemple le carrelage sur le sol, un mur de briques. Il est clair que cette définition n'est pas générale, car dans la nature beaucoup d'objet sont aléatoirement texturés (par exemple l'écorce des arbres, les feuillages d'une haie, autres), cette méthode n'est donc valable que pour des applications spécifiques.
2. **Méthodes fondées sur des modèles physiques ou templates.** Ces méthodes exploitent un modèle d'un motif-image appelé *template*, lequel sert à caractériser la texture ou à la synthétiser. Les paramètres de ces modèles doivent capturer les qualités essentielles qui caractérisent la texture perçue. Les champs aléatoires de Markov et les approches par *fractales* (autosimilarité à des échelles différentes) sont deux exemples typiques de ce type de représentation de la texture.

3. **Méthodes fondées sur un filtrage.** Ces méthodes exploitent le fait que le système visuel humain réalise des analyses fréquentielles de l'image, ce qui rend la texture spécialement adaptée à ce type d'analyse.
4. **Modèles statistiques.** Ces méthodes n'exploitent pas la notion de motif. Elles utilisent des attributs statistiques définis sur des pixels et leur voisinage comme primitives élémentaires de texture [61]. Ces textures peuvent avoir un aspect désordonné mais sont cependant considérées comme homogènes ce qui rend ce type de modèles très adapté aux textures réelles. Les modèles statistiques de texture les plus représentatifs, exploitent les populaires matrices de co-occurence, les coefficients d'auto-corrélation et la représentation que nous avons choisie, les histogrammes de sommes et différences.

Toutes ces méthodes sont utilisées sur les images de luminance et chacune d'elle a une définition particulière de la texture, qui est la base pour la définition de l'opérateur chargé de la mesurer. La définition la plus appropriée de la texture correspondant à la méthode que nous présentons et utilisons pour calculer les attributs de texture, c'est la définition donnée par Haralick [56] qui est :

Définition La texture est défie comme un modèle usuellement particulier et répété dans la surface d'un objet donné et qui est essentiellement considéré comme une propriété de voisinage.

Après avoir donné la définition particulière de la texture, nous présentons notre méthode, qui est détaillée dans la section suivante.

Histogrammes de sommes et différences

Les matrices de co-occurrence (MC) sont une des approches les plus utilisées pour l'analyse de texture dans des images. Différents attributs de texture statistiques sont calculés à partir des MC. Toutefois, les attributs ne sont pas complètement indépendants et la mémoire nécessaire est très grande. Par contre, une alternative pour les MC est l'usage des histogrammes de sommes et différences (HSD).

Ces histogrammes sont considérés comme une signature pour une image ou une région. M. Unser a donc proposé une méthode basée sur des histogrammes monodimensionnels pour estimer la texture sur une région d'une image [138]. Il est évident que la complexité algorithmique de cette approche est moindre que celle des MC. Ainsi, cette approche a besoin d'un espace mémoire réduit et c'est une excellente approximation des mesures proposées par Haralick [56].

Soit $K \times L$ une région rectangulaire qui contient l'image discrète d'une texture à analyser qui est désignée par $I(k,l)$, où $(k \in [0, K-1])$ et $(l \in [0, L-1])$. Supposons que le niveau de gris de chaque pixel est quantifié aux N_g niveaux, car soit $G \in [0, ..., N_g - 1]$ l'ensemble de ces niveaux N_g. Ensuite, pour un pixel donné (k,l), soit $(\delta_k, \delta_l) = \{(\delta_{k_1}, \delta_{l_1}), (\delta_{k_2}, \delta_{l2}), ..., (\delta_{k_M}, \delta_{l_M})\}$ l'ensemble de M déplacements relatifs. Les images de sommes et différences, I_S et I_D respectivement, sont associées à chaque déplacement relatif (δ_k, δ_l), de sorte que :

$$I_S(k,l) = I(k,l) + I(k+\delta_k, l+\delta_l) \tag{3.17}$$

$$I_D(k,l) = I(k,l) - I(k+\delta_k, l+\delta_l) \tag{3.18}$$

Ainsi, le rang de l'image I_S est $[0, 2(N_g-1)]$ et pour l'image I_D est $[-N_g+1, N_g-1]$. Or, soit i et j deux niveaux de gris quelconques dans les rangs des images I_S et I_D respectivement. Puis, soit D un sous-ensemble d'indices qui spécifie la région rectangulaire à analyser, ainsi, les HSD avec les paramètres (δ_k, δ_l) dans le domaine $(k,l) \in D$ sont, respectivement, définis comme :

$$h_S(i;\delta_k,\delta_l) = h_S(i) = \#\{(k,l) \in D, I_S(k,l) = i\} \tag{3.19}$$

$$h_D(j;\delta_k,\delta_l) = h_D(j) = \#\{(k,l) \in D, I_D(k,l) = j\} \tag{3.20}$$

où le nombre total de compte ou le nombre de points qui appartient à la région est obtenu par la relation suivante :

$$N = \#\{D\} = K \times L = \sum_i h_S(i) = \sum_j h_D(j) \tag{3.21}$$

Il est donc possible de caractériser un échantillon spécifique de texture par une collection d'histogrammes de sommes et différences estimés pour les différents déplacements relatifs (δ_k, δ_l). En outre, les histogrammes de sommes et différences normalisés peuvent être calculés par la relation suivante :

$$\widehat{P_S}(i) = \frac{h_S(i)}{N}, \qquad \widehat{P_S}(i) \in [0,1] \tag{3.22}$$

$$\widehat{P_D}(j) = \frac{h_D(j)}{N}, \qquad \widehat{P_D}(j) \in [0,1] \tag{3.23}$$

Ces histogrammes normalisés peuvent être interprétés comme suit : $P_S(i) = \widehat{P_S}(i)$ est la probabilité que la somme des pixels $I(k,l)$ et $I(k+\delta_k, l+\delta_l)$ aient la valeur i ; $P_D(j) = \widehat{P_D}(j)$ est la probabilité que la différence des pixels $I(k,l)$ et $I(k+\delta_k, l+\delta_l)$ aient la valeur j.

En utilisant ces probabilités, on dispose d'une caractérisation probabiliste de l'organisation spatiale des pixels dans une région basée sur une analyse de voisinage. Cependant, dans la plupart des applications réelles, il est nécessaire de réduire la dimension des attributs de la texture. Unser [138] a proposé une série de mesures pour extraire uniquement l'information de texture utile à partir des HSD. Les attributs les plus représentatifs dérivés de ces distributions sont résumés dans la table 3.1.

TABLE 3.1 – Attributs de texture calculés à partir des histogrammes de sommes et différences

Attributs	Formules de texture
Moyenne	$\mu = \frac{1}{2}\sum_i i \cdot \widehat{P_S}(i)$
Variance	$\frac{1}{2}\left(\sum_i (i-2\mu)^2 \cdot \widehat{P_S}(i) + \sum_j j^2 \cdot \widehat{P_D}(j)\right)$
Corrélation	$\frac{1}{2}\left(\sum_i (i-2\mu)^2 \cdot \widehat{P_S}(i) - \sum_j j^2 \cdot \widehat{P_D}(j)\right)$
Contraste	$\sum_j j^2 \cdot \widehat{P_D}(j)$
Homogénéité	$\sum_j \frac{1}{1+j} \cdot \widehat{P_D}(j)$
Cluster de Nuance	$\sum_i (i-2\mu)^3 \cdot \widehat{P_S}(i)$
Cluster de Proéminence	$\sum_i (i-2\mu)^4 \cdot \widehat{P_S}(i)$

Notons que la plupart de ces expressions sont très similaires aux attributs proposés par Haralick pour les MC mais elles sont obtenues avec une complexité algorithmique moindre. Notons égalemment que le calcul de ces attributs se fait pour des décalages δ_k, δ_l déterminés ; notre stratégie pour choisir ces décalages va dépendre de la phase pour laquelle ce calcul de texture est effectué.

Etant donné que les ressources de calcul et de mémoire sont limitées sur un système embarqué sur un robot, nous avons adopté l'approche par attributs de texture pour la caractérisation des régions

d'une image acquise sur une scène. C'est une méthode rapide qui permet d'extraire des descripteurs de texture avec une quantité limité de mémoire. Bien que ce soit une méthode rapide qui ne demande pas beaucoup de ressources, elle a été initialement conçue pour être mise en œuvre de façon séquentielle. Il est donc nécessaire d'en développer une application parallèle plus appropriée pour la mise en œuvre sur de matériel.

Adéquation des histogrammes de sommes et différences

Un des problèmes essentiels dans la mise en œuvre sur une architecture matérielle est l'adéquation algorithmique. Souvent, l'optimisation de l'algorithme en fonction de la technologie utilisée, permet de gagner en performance, ou de limiter les ressources en éléments logiques ou en mémoire. Nous avons dans un premier temps, conçu une architecture qui calcule les attributs à partir des HSD. La mise en œuvre de l'algorithme classique des HSD a besoin d'une quantité relativement importante de mémoire pour stocker les histogrammes. Cependant si pour une implémentation logicielle la quantité de mémoire nécessaire n'est pas une contrainte, elle peut le devenir pour une implémentation matérielle, surtout si la cible est de type FPGA car ceux-ci embarquent peu de mémoire. Pour contourner le problème il est alors fait appel à des circuits mémoires extérieurs ce qui ralentit les temps d'accès et réduit par voie de conséquence les performances temps réel.

Ainsi, pour contourner la difficulté, nous avons développé une adéquation algorithmique qui consiste à calculer les attributs de texture sans avoir besoin de calculer les histogrammes. Ceci nous permet de réduire d'une part les besoins en mémoire et, d'autre part la complexité de l'algorithme et, indirectement, l'optimisation du temps de calcul.

L'attribut de moyenne est donné par l'équation suivante :

$$\mu = \frac{1}{2} \sum_{i} i \cdot \widehat{P_S}(i) \tag{3.24}$$

où $\widehat{P_S}(i)$ pourrait être remplacé par les histogrammes de sommes normalisés donnés par l'équation 3.22

$$\mu = \frac{1}{2} \sum_{i} i \cdot \frac{h_S(i)}{N} = \frac{1}{2N} \sum_{i} i \cdot h_S(i) \tag{3.25}$$

où $h_S(i)$ représente les histogrammes de sommes dans la région rectangulaire, c'est à dire, le nombre de fois que le niveau de gris i est présent dans les images de sommes dans le domaine D. Par conséquent, le terme $i \cdot h_S(i)$ pourrait être remplacé dans l'équation 3.25 par $h_S(i)$ sommes d'i.

$$\mu = \frac{1}{2N} \sum_{i} \sum_{m=1}^{h_S(i)} i \tag{3.26}$$

En outre, en utilisant l'équation 3.19, il est possible de remplacer les sommations dans l'espace d'attributs par des sommations dans l'espace du domaine, en remplaçant i par $I_S(k,l)$ et en changeant par l'espace approprié.

$$\mu = \frac{1}{2N} \sum_{k \in D} \sum_{l \in D} I_S(k,l) \tag{3.27}$$

Ce résultat est l'équation classique pour le calcul de la moyenne. Toutefois, la contribution la plus importante concerne le changement de l'espace du domaine D, ce qui permet de calculer l'attribut de texture de moyenne sans calculer les histogrammes. Cette opération évite donc l'utilisation de mémoire supplémentaire pour stocker les histogrammes et réduit également la complexité de l'algorithme.

Prenons un exemple plus complexe : l'attribut de variance, lequel est donné par l'équation suivante :

$$\frac{1}{2}\left(\sum_i (i-2\mu)^2 \cdot \widehat{P_S}(i) + \sum_j j^2 \cdot \widehat{P_D}(j)\right) = \frac{1}{2}\sum_i (i-2\mu)^2 \cdot \widehat{P_S}(i) + \frac{1}{2}\sum_j j^2 \cdot \widehat{P_D}(j) \quad (3.28)$$

Remplaçons les histogrammes de sommes et différences normalisés donnés par les équations 3.22 et 3.23

$$\frac{1}{2}\sum_i (i-2\mu)^2 \cdot \frac{h_S(i)}{N} + \frac{1}{2}\sum_j j^2 \cdot \frac{h_D(j)}{N} = \frac{1}{2N}\sum_i (i-2\mu)^2 \cdot h_S(i) + \frac{1}{2}\sum_j j^2 \cdot h_D(j) \quad (3.29)$$

Changeons les multiplications par les histogrammes de sommes et différences ($h_S(i)$ et $h_D(j)$) par des sommations successives.

$$\frac{1}{2N}\sum_i \sum_{m=1}^{h_S(i)} (i-2\mu)^2 + \frac{1}{2N}\sum_j \sum_{m=1}^{h_D(j)} j^2 \quad (3.30)$$

Enfin, en changeant l'espace d'attributs par l'espace du domaine et en remplaçant i par $I_S(k,l)$ et j par $I_D(k,l)$ et en changeant par l'espace approprié, l'équation devient :

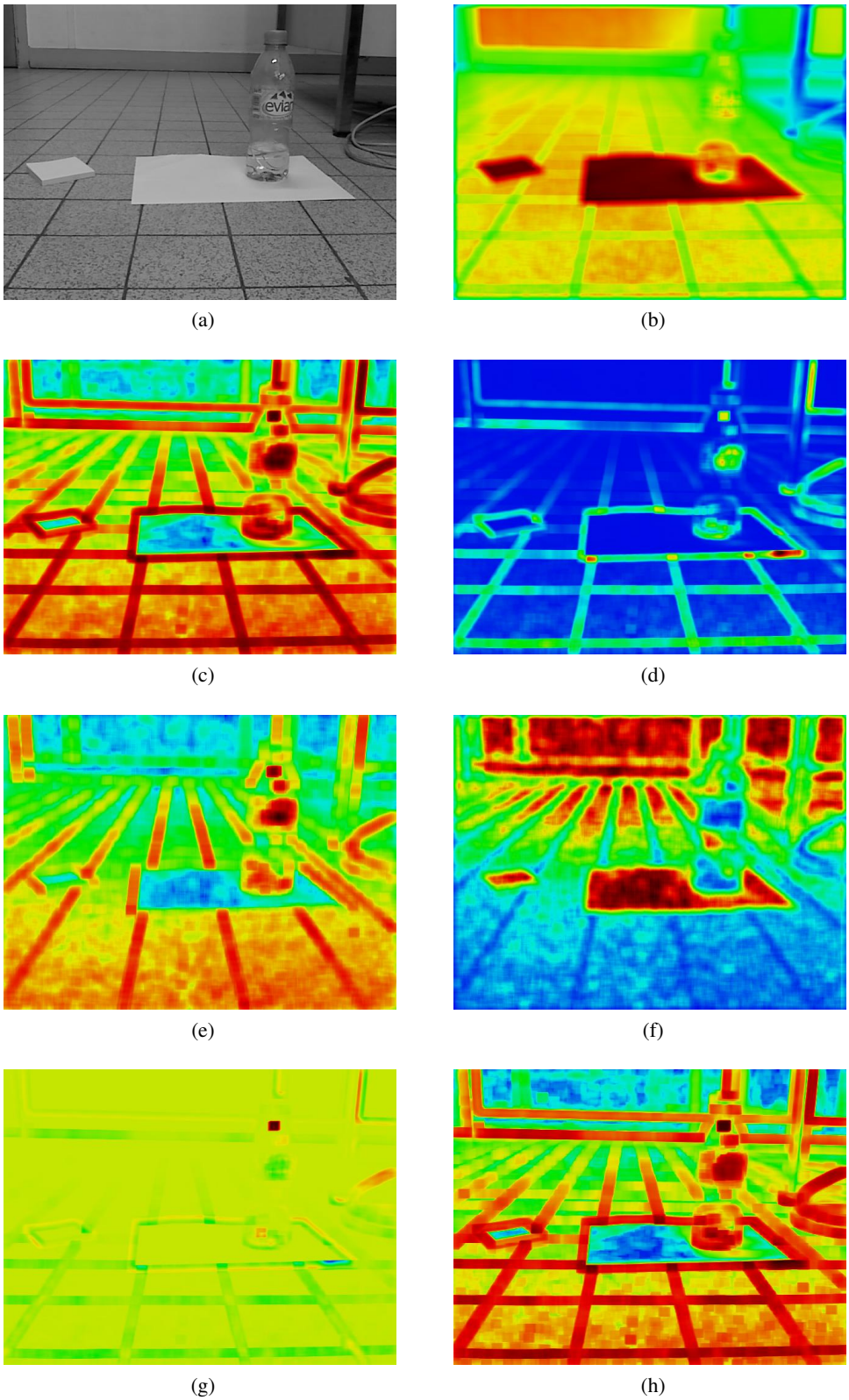
$$\frac{1}{2N}\left(\sum_k \sum_l (I_S(k,l)-2\mu)^2 + \sum_k \sum_l I_D(k,l)^2\right) \quad (3.31)$$

Les attributs de texture les plus couramment utilisés sont présentés dans le tableau 3.2. L'adéquation de chacun des attributs de texture a été analysée et calculée à partir des attributs classiques de HSD de la même façon que ceux qui ont été déjà démontrés. La figure 3.6 présente les attributs de texture normalisés calculés pour un déplacement de $(-1, 0)$.

TABLE 3.2 – Attributs de texture calculés à partir de l'adéquation des histogrammes de sommes et différences

Attributs	Formule de texture
Moyenne	$\frac{1}{2N}\sum_k \sum_l I_S(k,l)$
Variance	$\frac{1}{2N}\left(\sum_k \sum_l (I_S(k,l)-2\mu)^2 + \sum_k \sum_l I_D(k,l)^2\right)$
Corrélation	$\frac{1}{2N}\left(\sum_k \sum_l (I_S(k,l)-2\mu)^2 - \sum_k \sum_l I_D(k,l)^2\right)$
Contraste	$\frac{1}{N}\sum_k \sum_l I_D(k,l)^2$
Homogénéité	$\frac{1}{N}\sum_k \sum_l \frac{1}{1+I_D(k,l)^2}$
Cluster de Nuance	$\frac{1}{N}\sum_k \sum_l (I_S(k,l)-2\mu)^3$
Cluster de Proéminence	$\frac{1}{N}\sum_k \sum_l (I_S(k,l)-2\mu)^4$

Ainsi, cette approche représente une technique intéressante en termes de simplification et d'optimisation pour l'analyse des attributs de texture.



(a) (b) (c) (d) (e) (f) (g) (h)

FIGURE 3.6 – Calcul des attributs de texture pour un déplacement de $(-1, 0)$. (a) Image d'origine, Images des attributs de la texture de (b) Moyenne, (c) Variance, (d) Corrélation, (e) Contraste, (f) Homogénéité, (g) Cluster de Nuance, (f) Cluster de Proéminence.

3.2.4 Construction et analyse de la base d'apprentissage

Dans notre approche, nous avons opté pour l'utilisation des informations de couleur et de texture pour caractériser et classifier les objets. Pourquoi ce choix ? Afin de mieux les différencier, les attributs dans un espace hybride couleur-texture sont plus riches en information que chacune des composantes prise indépendamment. De plus ces attributs peuvent être renforcés avec des informations contextuelles comme la position et la forme.

La sélection pertinente des attributs de bas niveau (la couleur et la texture) est importante pour mieux caractériser les objets. L'ensemble des attributs doit être déterminé en fonction de son pouvoir discriminant. Il est aussi important de considérer l'invariance des attributs à plusieurs facteurs comme la rotation, la translation, les conditions d'illumination, les changements de points de vue, les occultations, la distance entre le capteur et la scène (échelle) entre autres.

Les défaillances d'invariance pourront être atténuées en réalisant un apprentissage qui prend en compte les caractéristiques de l'environnement. Néanmoins, cette solution n'est pas la panacée : pour éviter une explosion combinatoire soit au moment de l'apprentissage des attributs de chaque classe, soit au moment de la classification des pixels, il faut aussi limiter la taille de la base d'images.

Dans les sections 3.2.3 et 3.2.2, nous avons présenté les principales méthodes de représentation de la couleur et de la texture. Ceci va nous permettre de justifier les méthodes de couleur et de texture choisies pour notre application. Etant donné que le système doit être embarqué sur un robot muni d'une capacité limitée de stockage et de manipulation de données, les méthodes que nous avons choisies sont celles des attributs de l'espace de couleur *CIE-Lab* pour la couleur et des attributs calculés à partir de l'adéquation des histogrammes des sommes et différences pour la texture.

La couleur et la texture nous permettent de disposer d'un vecteur caractéristique représentatif pour chaque objet dans une scène. Ainsi, les attributs de texture sont calculés en choisissant les déplacements relatifs δ_k et δ_l selon les critères suivantes :

- Pour limiter la taille du vecteur caractéristique d'un objet.
- Pour limiter les ressources nécessaires dans la phase de classification.
- Pour augmenter le pouvoir de discrimination dans la phase de classification et d'apprentissage.

Nous calculons les sommes et différences pour seize déplacements relatifs dans le voisinage proche de chaque pixel $(\delta_k, \delta_l) = \{(-1,0), (-1,-1), (0,-1), (1,-1), (-2,0), (-2,-2), (0,-2), (2,-2), (-3,0), (-3,-3), (0,-3), (3,-3), (-4,0), (-4,-4), (0,-4), (4,-4)\}$ qui correspondent aux déplacements relatifs pour quatre distances (1, 2, 3 et 4 pixels) et quatre directions ($0°$, $45°$, $90°$ and $135°$). Ces déplacements seront analysés afin de découvrir s'ils nous donnent autant d'information utilisable pour la classification, réduisant ainsi la taille du vecteur caractéristique et par conséquent les ressources de calcul nécessaires.

L'algorithme détaillé de calcul des attributs de la couleur et de la texture est représenté dans la figure 3.7. Tout d'abord, nous transformons l'image *RGB* provenant de l'algorithme de démosaïquage, en une image en composantes primaires *XYZ*, afin de calculer l'espace de couleur *CIE-Lab*. Les composantes chromatiques *a* et *b* de l'espace de couleur *CIE-Lab* sont utilisées comme des attributs de couleur pour la classification. La luminosité *L** est utilisée pour calculer les images de sommes et de différences dans l'analyse de texture. Dans cette dernière nous utilisons seulement cinq des sept attributs, ceux-ci étant choisis pour limiter les ressources requises par notre application sur la plateforme embarquée. Les attributs de moyenne et contraste sont calculés directement depuis l'image de somme et de différence, respectivement. Les attributs de variance et de corrélation sont obtenus à partir de l'image de somme et des attributs de moyenne et de contraste. Enfin, l'attribut d'homogénéité est calculé à partir de l'attribut de contraste. Tous les attributs de texture et de couleur constituent l'espace des attributs utilisé pour la classification.

Nous avons choisi l'espace des attributs dans lequel chaque pixel sera représenté. Notre objectif est

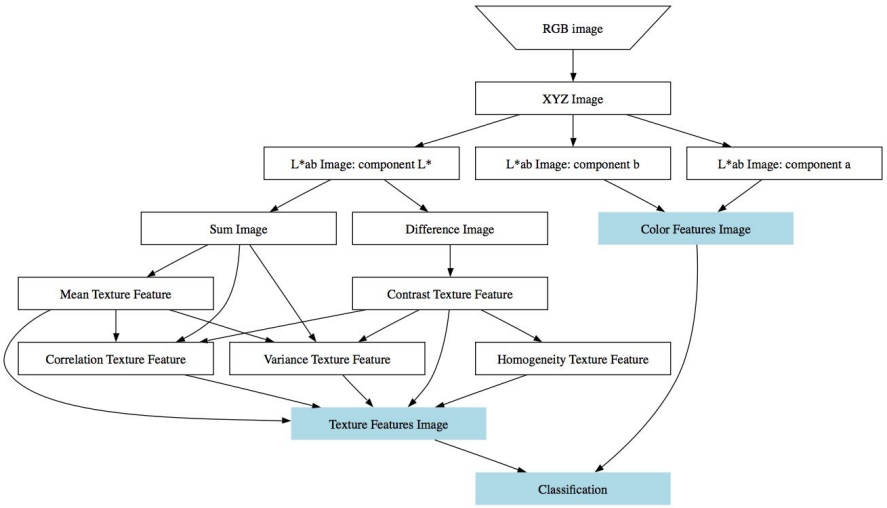


FIGURE 3.7 – Diagramme de l'algorithme pour le calcul des attributs de couleur et de texture.

d'étiqueter chaque pixel en fonction de la nature de l'objet de la scène dont elle est l'image. Nous devons apprendre, hors-ligne, les valeurs caractéristiques des attributs pour chaque classe des objets d'intérêt liés à notre application ; comme la classification implémentée dans ces travaux exploite des méthodes non paramétriques (k-PPV, SVM ou AdaBoost), il s'agit en fait d'apprendre un grand nombre d'échantillons pour chaque classe.

La base de données a été construite en mode supervisé ce qui a nécessité la détermination préalable des classes d'objet pertinentes pour l'application visée. Le nombre de classes est fixé à seulement deux parce qu'il s'agit de la détection d'un objet particulier dans la scène et non l'identification de chacun des objets de la scène. La base de données a été créée à partir d'images d'échantillon de certains obstacles et certaines parties du sol. La figure 3.8 montre quelques exemples de la base d'images utilisée pour créer la base de données. Un fichier contient les échantillons dans l'espace des attributs de couleur et texture calculés depuis la base d'images. Chacun des échantillons est marqué avec une étiquette indiquant s'il appartient à la classe d'intérêt (du sol) ou non (des obstacles). Le défi lors de l'apprentissage consiste à maximiser les informations utiles, tout en minimisant la taille de la base de données (évitant ainsi le problème de sur-apprentissage, de temps d'accès et de calcul).

3.2.5 Classification par attributs de couleur et de texture

Après avoir subi le pré-traitement évoqué dans la section 3.2.1, les images couleur dans l'espace *RGB* sont transformées dans l'espace de couleur *CIE-Lab*, les composantes chromatiques a et b sont utilisées dans le vecteur caractéristique, tandis que la luminosité L^* est utilisée pendant l'analyse de texture pour obtenir le reste des éléments du vecteur caractéristique. Ce vecteur est utilisé pour caractériser les objets dans la scène, et ainsi donc détecter les obstacles.

Les méthodes de reconnaissance utilisent l'information contenue dans une base d'apprentissage dans laquelle se trouvent des échantillons représentatifs étiquetés de chaque classe pour différents objets. Cette base est construite en mode supervisé. A partir de l'espace des attributs, une méthode de classification détermine la classe à laquelle correspond chaque pixel extrait d'une image ce qui rend possible l'identification de chaque objet dans la scène. La figure 3.9 montre les nuages de points correspondant aux deux classes dans la base de données, les étoiles rouges représentent la classe d'intérêt, à savoir, le sol, tandis que les cercles rouges représentent la classe correspondant à des obstacles. Les attributs de texture ont été

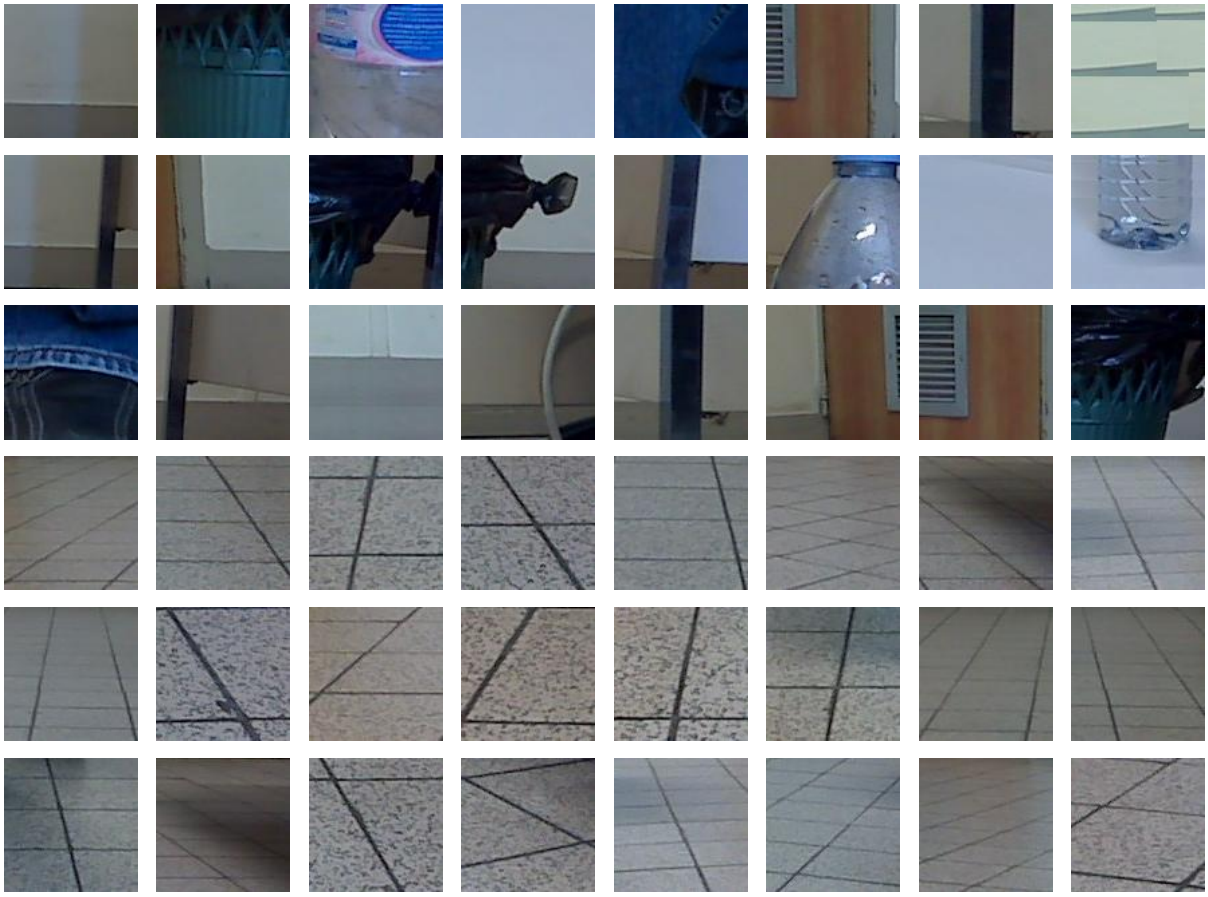
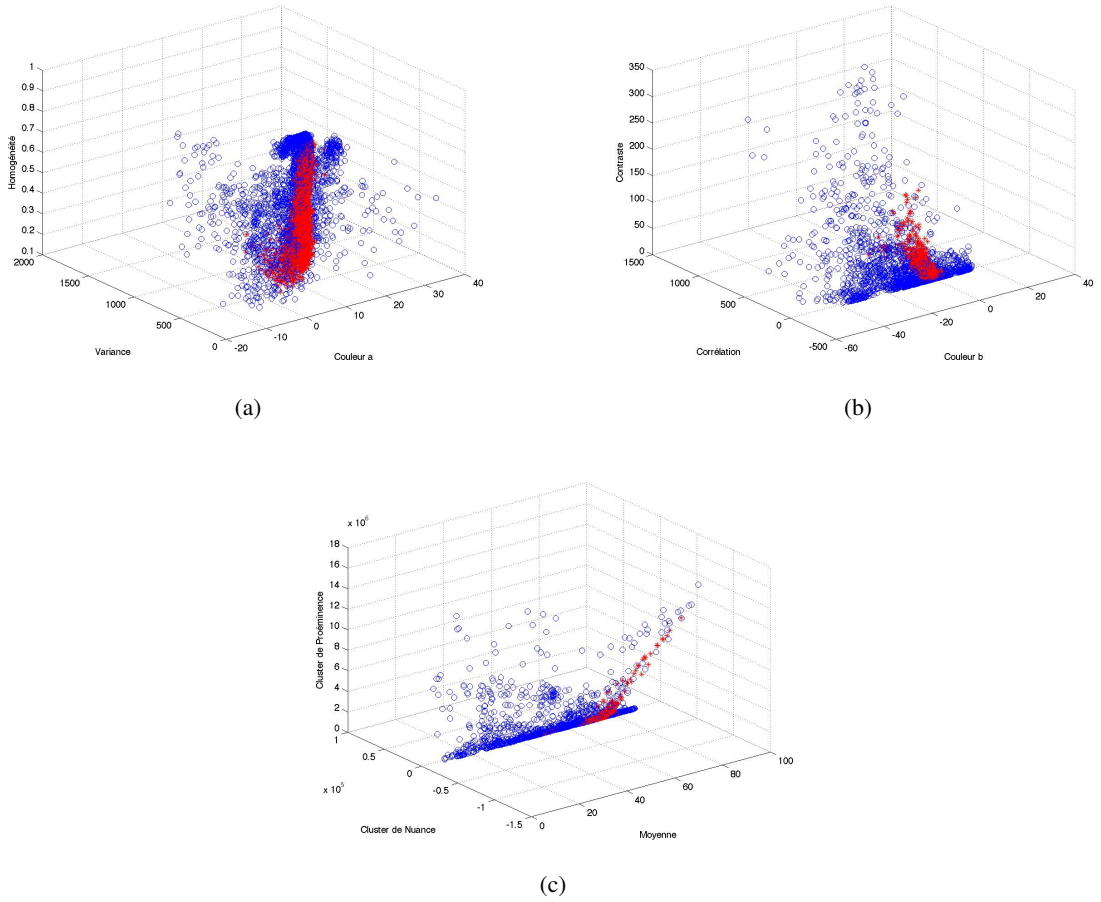


FIGURE 3.8 – Base d'images pour la phase d'apprentissage : les trois premières lignes correspondent aux obstacles et les trois derniers au sol.

calculés pour un déplacement relatif de $(-1, 0)$. Les trois graphiques suivant représentent chacun trois des neuf attributs de la couleur et de la texture. On peut voir que la classe des obstacles est la classe la plus hétérogène (grande variance sur les attributs) des deux : plus la classe est hétérogène, plus il faudra d'échantillons pour la représenter et plus il y aura des risques d'ambiguïtés inter classes. Ces graphiques montrent aussi la complexité et la non-linéarité de la classification nécessaires pour arriver à séparer les classes.

Dans la suite, nous décrirons brièvement les trois approches de classification qui ont été implémentées et évaluées par logiciel : *K-plus proches voisins* (k-PPV), *Support Vector Machines* (SVM) et *Adaptive Boosting* (AdaBoost).

– **K-plus proches voisins.** La méthode des k-PPV est particulièrement classique. Elle offre l'avantage d'être une technique très simple mais puissante dédiée principalement à la classification et qui peut être étendue à des tâches d'estimation. Elle s'appuie sur le principe de prise de décision en mesurant la similarité d'un élément inconnu avec des échantillons déjà stockés et classés. Ainsi, on doit simplement chercher les étiquettes de classe d'un certain nombre k de *plus proches voisins* et en fonction de ces voisins, prendre une décision pour assigner une étiquette à cet élément inconnu. Cette méthode diffère des méthodes traditionnelles d'apprentissage car aucun modèle n'est appris à partir des échantillons. Les données après normalisation, sont simplement stockées en mémoire. Le nombre de plus proche voisins, k, doit être impair afin d'éviter les ambiguïtés et doit être maintenu petit car un grand nombre de k tend à créer des fausses classifications surtout si les



(a) (b)

(c)

FIGURE 3.9 – Nuages de points de la classe obstacle (cercles bleus) et sol (étoiles rouges).

classes individuelles ne sont pas bien séparées. Il est facile de vérifier que le résultat global obtenu par ce type de classificateur est toujours au moins la moitié de celui obtenu par le meilleur classificateur pour un problème donné. Par contre, un des inconvénients principaux de ces méthodes se situe dans l'étape d'apprentissage, c'est à dire, celle où le classificateur a besoin de toutes les données disponibles. Ceci peut entraîner une augmentation de la complexité de calcul et donc, du temps de classification, si la base de données est considérablement grande. En effet, contrairement aux autres méthodes de classification (SVM, arbres de décision, AdaBoost, réseaux de neurones, algorithmes génétiques, etc), il n'existe pas d'étape proprement dite d'apprentissage consistant à construire un modèle compact à partir des échantillons de la base de données.

– **Support Vector Machines.** Depuis quelques années, les SVM ont émergé comme une nouvelle technique d'apprentissage supervisé pour la classification de données. Boser, Guyon et Vapnik [15] ont proposé le premier classificateur binaire de cette catégorie qui a par ailleurs été étendu à des problèmes de régression. La méthode SVM est basée sur le principe de la minimisation de la fonction de risque structurel [139]. Cette méthode a pour objectif de rechercher le meilleur hyperplan($\mathbf{w^T} \cdot \mathbf{x} + \mathbf{b}$) de séparation des données en deux classes [83]. La classification d'un nouvel individu $\mathbf{X}$ de $\mathbb{R}^n$ est donnée par sa position par rapport à cet hyperplan, en fait par le signe obtenu en substituant $\mathbf{x}$ par l'échaantillon à classer $\mathbf{X}$ dans l'équation de l'hyperplan. Les solutions obtenues par SVM sont indépendantes de la dimension des données et ne tombent jamais dans des minimums locaux.

– **Adaptive Boosting.** La pondération de la performance d'une méthode a permis de développer des modèles particuliers qui gèrent bien les situations ou cas problématiques (ou aberrants) pour les méthodes classiques. Ainsi, il existe plusieurs variations de la méthode AdaBoost considérées comme des techniques d'apprentissage avancé utilisées dans l'optimisation des classificateurs. Cette méthode est non seulement la plus pratique, mais également elle repose sur des propriétés théoriques solides. La mise à jour adaptative de la distribution des échantillons, visant à augmenter le poids de ceux qui ont été mal appris par le classificateur précédent, permet d'améliorer les performances des algorithmes d'apprentissage. AdaBoost peut être très efficace avec des données de dimensions élevées en gérant plusieurs classificateurs dits «faibles» de telle manière qu'il se complètent [8].

On a évalué par simulation logicielle les trois approches dans les mêmes conditions pour le calcul des attributs de couleur et de texture. Les conditions, les paramètres et les résultats seront détaillés dans la section suivante. Les résultats de l'évaluation nous a permis de trouver le classificateur le plus approprié à notre problème en termes de performance. L'approche k-PPV a obtenu un pourcentage de reconnaissance de $93,65\%$, tandis que l'approche SVM atteint un pourcentage de $92,22\%$, $95,01\%$ et $93,67\%$ pour les noyaux (kernels) linéaire, quadratique et cubique, respectivement. Le meilleur pourcentage de reconnaissance appartient à l'approche AdaBoost avec un pourcentage de $96,76\%$. En conclusion, sur la base de ces résultats, la méthode la plus appropriée pour la classification est l'algorithme AdaBoost car c'est lui qui présente la meilleure performance pour la détection d'obstacles dans un environnement d'intérieur en utilisant les attributs de couleur et de texture. De plus cette approche demande beaucoup moins de mémoire que la méthode de k-PPV, ce qui est un aspect très important dans les implémentations sur du matériel. Rappelons aussi que les méthodes doivent donner une probabilité d'appartenance d'un échantillon aux différentes classes, ce qui permet de détecter des situations d'ambiguïtés et ce que la méthode SVM n'a pas. Pour toutes ces raisons nous avons choisi l'algorithme AdaBoost comme classificateur pour la détection d'obstacles. Nous détaillerons donc par la suite cet algorithme afin de bien comprendre son fonctionnement et sa mise en œuvre sur une cible matérielle.

Algorithme Adaptive Boosting

L'algorithme AdaBoost a été développé en 1995 par Freund et Schapire [45]. C'est un concept puissant d'apprentissage, qui donne une solution à la tâche de classification par apprentissage supervisé. Il combine la performance de plusieurs classificateurs dits «faibles» pour produire un puissant «consensus », connu comme classificateur fort. Freund et Schapire ont démontré que si chaque classificateur faible est juste un peu mieux que le hasard, l'erreur d'entraînement du classificateur final diminue exponentiellement. Le classificateur faible peut donc être très simple et avoir un temps de calcul très court. Plusieurs classificateurs faibles intelligemment combinés produisent un classificateur fort qui surpasse le plus souvent la performance perçue par les classificateurs «monolithiques» forts tels que SVM et réseaux de neurones.

Les arbres de décision sont les classificateurs faibles les plus couramment utilisés dans la technique de renforcement (connu comme boosting). Souvent, les arbres de décision plus simples avec seulement un nœud par arbre (appelé souche) sont suffisants. L'algorithme AdaBoost permet de résoudre de nombreuses difficultés pratiques des algorithmes de boosting.

L'objectif de l'algorithme est de trouver au moins la fonction dans l'espace d'hypothèses qui donne la plus petite erreur relative à la distribution D, laquelle distribution est délivrée à partir des échantillons étiquetés depuis la base d'apprentissage. L'algorithme AdaBoost a principalement deux différences par rapport à d'autres techniques d'apprentissage automatique :

1. L'algorithme AdaBoost ajuste ou adapte les erreurs dans les hypothèses données par les classificateurs faibles.

2. La limite de l'actionnement de l'algorithme AdaBoost dépend seulement de l'actionnement du modèle d'apprentissage faible qui est généré par la fonction de distribution sur le processus d'apprentissage.

L'idée de base de l'algorithme AdaBoost est la suivante : l'algorithme prend comment entrée un ensemble d'échantillons d'apprentissage $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$, où chaque x_i appartient à un domaine ou à l'espace X, et chaque étiquette y_i est un élément de l'ensemble Y. Pour notre travail, nous supposons que $Y = \{-1, +1\}$, que nous appelons la classe négative et la classe positive, respectivement. L'algorithme AdaBoost utilise un algorithme d'apprentissage faible comme classificateur de base, qui est utilisé à plusieurs reprises dans une série de $t = 1, ..., T$ itérations. L'un des objectifs principaux de l'algorithme est de maintenir une distribution D ou un ensemble de poids sur l'ensemble des échantillons d'apprentissage. Le poids de cette distribution pour l'échantillon d'apprentissage i dans l'itération t est dénotée $D_t(i)$. Au début, tous les poids sont égaux et uniformes (voir équation 3.32), mais à chaque itération, les poids des échantillons d'apprentissage mal classés sont augmentés de telle sorte que le classificateur faible suivant est obligé de se concentrer plus sur les échantillons difficiles à classer, et inversement lorsque les échantillons d'apprentissage sont bien classés.

$$D_1(i) = \frac{1}{m} \quad \forall i \tag{3.32}$$

Le travail de l'apprentissage faible consiste à s'entraîner sur les m échantillons d'apprentissage pour retrouver la meilleure des hypothèses faibles (classificateur faibles) $h_t : X \rightarrow \{-1, +1\}$ appropriée pour la distribution de poids D_t. Cette opération est répétée T fois sur les mêmes échantillons, mais à chaque fois la distribution D_t change ; de fait, les hypothèses trouvées à chaque itération vont aussi changer. Ces hypothèses peuvent se voir comme des attributs qui permettent de différencier les deux classes avec une erreur plus petite que le hasard, donc $\epsilon_t > 1/2$. Cette erreur est calculée comme la somme des poids des échantillons qui ont été mal classés. Une fois que l'hypothèse faible h_t a été choisie, l'algorithme AdaBoost calcule intuitivement un paramètre α_t, lequel mesure l'importance qui est assignée à l'hypothèse faible h_t. L'hypothèse finale H (voir équation 3.33) est construite comme une combinaison des hypothèses faibles trouvées à chaque itération. L'hypothèse finale produite par l'algorithme AdaBoost comporte des erreurs $\epsilon 1, \epsilon 2, ...\epsilon T$, où l'erreur ϵ_t pour l'itération t est définie par l'equation 3.34. Cette erreur permet de mettre à jour :

- Le coefficient α_t de l'hypothèse faible h_t.
- Les nouveaux poids des échantillons d'apprentissage (D_t).

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{3.33}$$

$$\epsilon_t = Pr_{D_t}\left[h_t(x_i) \neq y_i\right] \tag{3.34}$$

Le classifieur AdaBoost est présenté dans l'algorithme 1. Dans la phase de classification, un nouvel échantillon est classifié en utilisant l'hypothèse finale forte, qui combine N hypothèses faibles apprises hors ligne à partir de la base d'apprentissage. Toutes les hypothèses faibles $h_1, h_2, ..., h_T$ apprises peuvent faire la classification de cet échantillon en parallèle ; le résultat retourné par h_i est affecté par le poids correspondant α_i. La combinaison linéaire de ces résultats donnera la probabilité d'appartenance de l'échantillon à la classe. Cette probabilité est positive lorsque l'échantillon appartient à la classe et négative dans le cas contraire. Ce résultat fournira donc la classe qu'on assignera à l'échantillon, soit $+1$ ou -1. Cette méthode peut se voir comme un vote majoritaire, la classe qui aura le plus de votes, sera la classe choisie pour l'échantillon à classifier.

Algorithme 1 Adaptive Boosting

Require: $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$, où $x_i \in X \wedge y_i \in Y = \{-1, +1\}$
$D_1(i) \Leftarrow \frac{1}{m}$
for $t = 1, 2, ..., T$ **do**
$h_t : X \rightarrow \{-1, +1\}$ {Entraîner les hypothèses faibles sur la distribution D_t}
$\epsilon_t \Leftarrow Pr_{D_t}[h_t(x_i) \neq y_i] = \sum_i D_t(i)\ \forall h_t(x_i) \neq y_i$ {Calculer l'erreur d'apprentissage}
$h_t \Leftarrow h_t|min(\epsilon_t)$ {Trouver l'hypothèse faible h_t ayant l'erreur d'apprentissage minimale ϵ_t sur la distribution D_t}
$\beta_t \Leftarrow \frac{\epsilon_t}{1-\epsilon_t}$
$\alpha_t \Leftarrow -\frac{1}{2} ln(\beta_t)$ {Calculer le coefficient de l'hypothèse faible h_t}
$D_{t+1}(i) \Leftarrow \frac{D_t(i)}{Z_t} \times exp(-\alpha_t y_i h_t(x_i))$ {Calculer la fonction de distribution D_{t+1}, où z_t est un facteur de normalisation}
end for
$H(x) \Leftarrow sign(\sum_t \alpha_t h_t(x))$ {Obtenir l'hypothèse finale}

L'algorithme AdaBoost est facile à mettre en œuvre. Il n'a besoin que d'un seul paramètre à régler, le nombre d'hypothèses faibles T, qui dépend directement du seuil d'erreur que l'on veut atteindre. Il ne requiert pas de caractéristiques spéciales pour les hypothèses faibles, donc il peut être combiné avec n'importe quelle méthode pour les trouver. Cette méthode a été utilisée jusqu'à présent dans de nombreuses applications avec un temps de calcul et un pourcentage de reconnaissance très satisfaisants.

Un inconvénient toutefois : l'algorithme AdaBoost est sensible au bruit. Cela peut avoir des effets pervers : l'algorithme va se concentrer sur plusieurs échantillons difficiles à classifier dans une zone d'ambiguïté inter classes ; ceci peut affecter négativement la performance du classificateur fort. Il faut donc faire attention à la base d'apprentissage qu'on lui donne.

Nous avons choisi comme classificateur faible la méthode des arbres de décision plus particulièrement des arbres de régression et classification (CART : Classification and Regression Tree). Cette méthode a l'avantage d'être facile à mettre en œuvre sur du matériel pour la classification, ce qui justifie ce choix pour l'implémentation finale sur une plateforme dédiée reprogrammable. De plus, la performance donnée par cette méthode est suffisante pour être combinée avec l'algorithme AdaBoost. Nous décrirons la méthode d'arbre de classification plus en détail dans la section suivante.

Arbres de décision : Arbres de régression et classification

Un arbre de décision est un modèle prédictif qui est utilisé pour représenter des modèles de classification et de régression. Un arbre de décision est appelé arbre de classification quand il est utilisé pour classer un objet ou une instance dans un ensemble de classes prédéfinies en fonction des valeurs de leurs attributs.

Un arbre de classification est une séquence ordonnée de questions, la question posée à chaque étape de la séquence dépendant des réponses aux question précédentes. La séquence se termine par une prédiction de la classe. Le point unique de départ d'un arbre de classification est appelé *nœud racine* et se compose de l'ensemble des échantillons d'apprentissage m au sommet de l'arbre. Un *nœud* est un sous-ensemble de l'ensemble des variables, et il peut être un nœud terminal ou non-terminal.

Un *nœud non-terminal* (ou de décision ou parent) est un nœud qui se divise en deux nœuds successeurs (une division binaire). Une telle division binaire est déterminée par une condition booléenne entre la valeur d'une variable unique et la valeur observée de cette variable, ce qui se traduit par une condition satisfaite («oui») ou pas («non»). Tous les échantillons qui ont atteint un nœud (parent) particulier et satisfont à la condition pour cette variable doivent descendre par un des deux nœuds successeurs ; les autres

échantillons qui ont atteint ce nœud (parent)et qui ne remplissent pas la condition doivent descendre par l'autre nœud successeur.

Un nœud qui n'est pas divisé est appelé un *nœud terminal* (ou feuille) et il va attribuer une étiquette indiquant la classe. Chaque échantillon de l'ensemble d'apprentissage m est analysé dans l'un des nœuds terminaux. Lorsqu'un nouvel échantillon de classe inconnue veut être analysé, on «lâche» celui-ci dans l'arbre jusqu'à ce qu'il atteigne un nœud terminal : la classe attribuée à cet échantillon correspond à l'étiquette attachée à ce nœud terminal. Il peut y avoir plus d'un nœud terminal avec la même étiquette de classe. Un arbre de division simple avec seulement deux nœuds terminaux est appelé une souche. L'ensemble des nœuds terminaux est appelé une *partition* de données. Quand l'échantillon a été lâché vers tous les chemins menant à un nœud terminal pouvant prédire une classe et en considérant simplement un «vote» de tous les échantillons d'apprentissage, la classe avec le plus grand nombre de votes est la classe que nous aurions prédite pour le nouvel échantillon.

L'algorithme de *partitionnement récursif* est la clé de la méthode statistique non-paramétrique des arbres de classification et régression [16]. Le partitionnement récursif est le processus étape par étape par lequel un arbre de décision se construit par la division ou non de chacun des nœuds de l'arbre en deux nœuds successeurs. Une caractéristique intéressante de la méthode CART est le fait que l'algorithme de classification consiste en une séquence hiérarchique de questions booléennes (par exemple, est-ce $x_i \geq \theta_j$?, où θ_j est une valeur de seuil), il est relativement simple à comprendre, à mettre en œuvre et pour interpréter les résultats.

La méthode CART est une technique d'apprentissage supervisée, la méthode prend comme entrée un ensemble d'échantillons d'apprentissage $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$, où chacun x_i appartient à un domaine X dans l'espace $\mathbb{R}^r$, et chaque étiquette y_i est un élément de l'ensemble Y. Dans la méthode CART, l'espace d'entrée $\mathbb{R}^r$ est divisé en un certain nombre de régions rectangulaires ($r = 2$) ou cuboïdes ($r > 2$) qui ne se chevauchent pas et dont chacune de régions est considérée comme homogène dans le but de prédire y_i. Chaque région a ses côtés parallèles aux axes de l'espace d'entrée et se voit attribuer une classe. Une telle partition correspond à un arbre de classification.

Considérons un exemple simple de partitionnement récursif impliquant deux variable d'entrée, X_1 et X_2. Supposons que le diagramme de l'arbre de décision soit donné dans la figure 3.10a. Les étapes possibles de cet arbre sont les suivantes :

1. Est-ce $X_1 \geq \theta_1$?, Si la réponse est oui, on suit la branche de gauche ; sinon, on suit la branche de droite.
2. Si la réponse à (1) est oui, alors on pose la question suivante : Est-ce $X_2 \geq \theta_2$?, une réponse affirmative nous donne la voie au nœud terminal τ_1 avec la région correspondant à $R_1 = \{X_1 \geq \theta_1, X_2 \geq \theta_2\}$; une réponse négative nous donne la voie au nœud terminal τ_2 avec la région correspondant à $R_2 = \{X_1 \geq \theta_1, X_2 < \theta_2\}$.
3. Si la réponse à (1) est non, alors on pose la question suivante : Est-ce $X_1 \geq \theta_3$?, si la réponse à (3) est oui, alors on pose la question suivante : Est-ce $X_2 \geq \theta_4$?, une réponse affirmative nous donne la voie au nœud terminal τ_3 avec la région correspondant à $R_3 = \{X_1 < \theta_1, X_1 \geq \theta_3, X_2 \geq \theta_4\}$; sinon, on suit la branche de droite au nœud terminal τ_4 avec la région correspondant à $R_4 = \{X_1 < \theta_1, X_1 \geq \theta_3, X_2 < \theta_4\}$.
4. Si la réponse à (3) est non, on arrive au nœud terminal τ_5 avec la région correspondant à $R_5 = \{X_1 < \theta_3\}$.

Nous avons supposé que $\theta_1 > \theta_3$ et $\theta_4 > \theta_2$. La division résultant des cinq régions dans l'espace d'entrée $\mathbb{R}^2$ est donnée dans la figure 3.10b. Pour un arbre de classification, chaque nœud terminal et région correspondante se voientt attribuer une étiquette indiquant la classe. Il est utile de noter que le type d'arbres développés par CART (appelés arbres binaires) ont la propriété que le nombre de nœuds terminaux est exactement un de plus que le nombre de nœuds de décision.

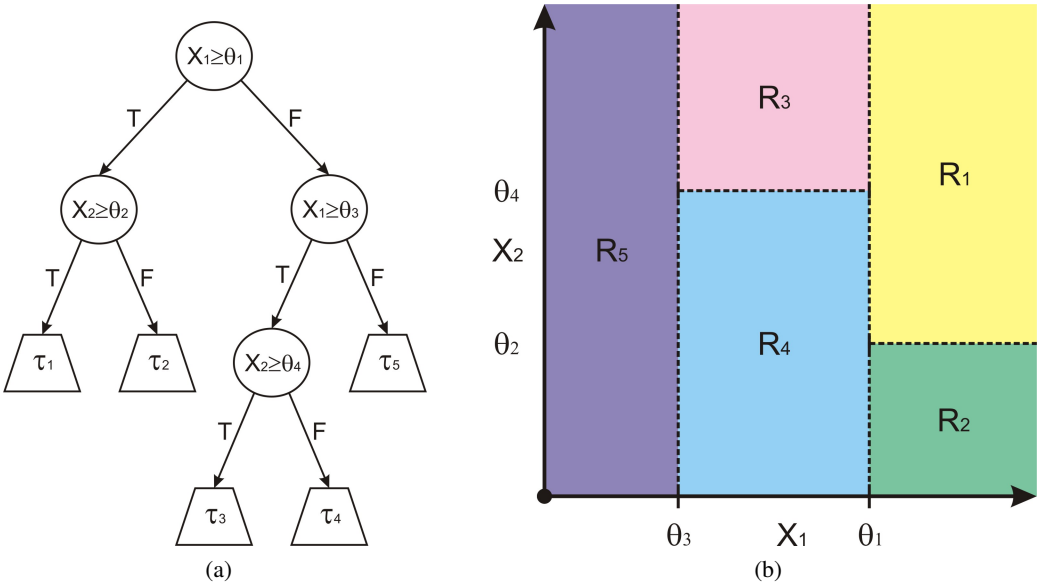


FIGURE 3.10 – Exemple de partitionnement récursif avec deux variables d'entrée X_1 et X_2. (a) Diagramme de l'arbre de décision avec cinq nœuds terminaux ($\tau_1 - \tau_5$) et quatre nœuds de décision correspondant aux quatre divisions dans l'espace d'entrée $\mathbb{R}^2$. (b) Cloisonnement de l'espace $\mathbb{R}^2$ dans cinq régions ($R_1 - R_5$) correspondant aux cinq nœuds terminaux.

La procédure pour faire pousser un arbre de classification ou algorithme d'arboriculture est la méthode utilisée pour entraîner les arbres de classification. La première étape consiste à choisir la «meilleure» variable de division dans l'espace d'entrée $\mathbb{R}^r$ pour chaque nœud. Il faut d'abord envisager toutes les divisions possibles sur toutes les variables présentes dans ce nœud, évaluer chacun d'eux et décider qui est le meilleur dans un certain sens. Les divisions sont simplement les points milieux entre deux paires de valeurs consécutives pour chaque variable.

Pour choisir la meilleure division sur toutes les variables, il faut d'abord choisir la meilleure division d'une variable donnée. Par conséquent, il faut définir une mesure de comparaison pour les divisions. La fonction d'«impureté» de nœud est la mesure d'hétérogénéité de composition dans une division et c'est elle qui fixe la mesure de comparaison pour les divisions.

Soit $\Pi_1, \Pi_2 ..., \Pi_K$ les $K \geq 2$ classes. Pour un nœud τ, nous définissons la fonction d'impureté $i(\tau)$ comme :

$$i(\tau) = \phi\left(p(1|\tau), p(2|\tau) ..., p(K|\tau)\right) \tag{3.35}$$

où $p(k|\tau)$ est une estimation de $P(x_i \in \Pi_k|\tau)$, la probabilité conditionnelle qu'un échantillon x_i est dans Π_k étant donné qu'il tombe dans le nœud τ. Dans l'équation 3.35 on a besoin que la fonction ϕ soit symétrique, définie sur l'ensemble de tous les K-uples de probabilités $(p_1, p_2, ..., p_k)$ avec la somme unité, minimisée en les points $(1, 0, ..., 0), (0, 1, ..., 0), ..., (0, 0, ..., 1)$ et maximisée au point $\left(\frac{1}{K}, \frac{1}{K}, ..., \frac{1}{K}\right)$. Dans le cas de deux classes ($K = 2$), les conditions sont réduites à une symétrie $\phi(p)$ avec un point maximum en $p = 1/2$ avec $\phi(0) = \phi(1) = 0$.

Plusieurs fonctions ont été suggérées, nous avons utilisé *l'indice de diversité de Gini*,

$$i(\tau) = \sum_{k \neq k'} p(k|\tau)\, p\left(k'|\tau\right) = 1 - \sum_{k} \left\{p(k|\tau)\right\}^2 \tag{3.36}$$

Dans le cas de deux classes, l'indice de Gini est réduit à

$$i(\tau) = 2p(1-p) \tag{3.37}$$

Supposons que nous voulions faire une division s dans le nœud τ de telle sorte qu'une proportion p_L des échantillons descende vers le nœud successeur de gauche τ_L et la proportion restante p_R des échantillons descende vers le nœud successeur de droite τ_R. Par exemple, supposons que nous ayons un ensemble d'échantillons dans lequel la variable de réponse $y_i \in Y$ a deux valeurs possibles, $Y = \{0, 1\}$. Supposons que l'une des divisions possible de la variable d'entrée X_j est $X_j \geq c$ contre $X_j < c$, où c est une valeur de X_j. La table 3.3 présente les estimations des proportions des échantillons dans la division par classe.

TABLE 3.3 – Table de deux par deux pour une division dans la variable X_j avec deux classes.

	$Y = 1$	$Y = 0$	# de la ligne
$X_j \geq c$	$n_{11} = \#\{x \vert X_j \geq c \& Y = 1\}$	$n_{12} = \#\{x \vert X_j \geq c \& Y = 0\}$	$n_{1+} = n_{11} + n_{12}$
$X_j < c$	$n_{21} = \#\{x \vert X_j < c \& Y = 1\}$	$n_{22} = \#\{x \vert X_j < c \& Y = 0\}$	$n_{1+} = n_{21} + n_{22}$
# de la colonne	$n_{+1} = n_{11} + n_{21}$	$n_{+2} = n_{12} + n_{22}$	$n_{++} = \#x = m$

Tout d'abord, on considère le nœud parent τ, on utilise la fonction de l'indice de Gini 3.36 comme mesure de l'impureté. On estime p_L par n_{+1}/n_{++} et p_R par n_{+2}/n_{++}, puis la fonction d'impureté est estimée à

$$i(\tau) = 1 - \left(\frac{n_{+1}}{n_{++}}\right)^2 - \left(\frac{n_{+2}}{n_{++}}\right)^2 \tag{3.38}$$

La fonction $i(\tau)$ est complètement indépendante de la division proposée. Ensuite, pour les nœuds successeurs, τ_L et τ_R. Pour la division $X_j \geq c$, on estime p_L par n_{11}/n_{1+} et p_R par n_{12}/n_{1+}, et pour $X_j < c$, on estime p_L par n_{21}/n_{2+} et p_R par n_{22}/n_{2+}, puis les fonctions d'impuretés sont estimées comme

$$i(\tau) = 1 - \left(\frac{n_{11}}{n_{1+}}\right)^2 - \left(\frac{n_{12}}{n_{1+}}\right)^2 \tag{3.39}$$

$$i(\tau) = 1 - \left(\frac{n_{21}}{n_{2+}}\right)^2 - \left(\frac{n_{22}}{n_{2+}}\right)^2 \tag{3.40}$$

La qualité de la division s dans le nœud τ est donnée par la réduction de l'impureté acquise par la division du nœud parent τ dans les deux nœuds successeurs, τ_L et τ_R.

$$\Delta i(s, \tau) = i(\tau) - p_L i(\tau_L) - p_R(\tau_R) \tag{3.41}$$

La meilleure division pour la seule variable X_j est celle qui a la plus grande valeur de $\Delta i(s, \tau)$ sur tous les $s \in S_j$, où S_j est l'ensemble des divisions distinctes possibles dans le nœud pour la variable X_j.

Afin de faire pousser un arbre, nous commençons avec le nœud racine qui se compose de l'ensemble des échantillons d'apprentissage. En utilisant le critère de qualité de division pour une seule variable, l'algorithme trouve la meilleure division pour le nœud racine pour chaque variable, X_1 à X_r. La meilleure division s dans le nœud racine est définie comme celle qui a la plus grande valeur pour l' équation 3.41 sur toutes les r meilleures divisions des variables simples dans le nœud. Ensuite, on fait la division de

chaque successeur du nœud racine de la même façon. On répète les divisions pour les nœuds qui suivent. Ce processus de division séquentielle pour la construction d'un arbre couche par couche est appelé *partitionnement récursif*. Si l'arbre de classification est cultivé jusqu'à ce qu'aucun des nœuds ne puissent plus être divisé, on dit que l'arbre est saturé. Il est très facile dans les problèmes ayant un grand nombre d'échantillons de faire pousser l'arbre considérablement, surtout si celui-ci est autorisé à croître jusqu'à saturation.

Une façon d'éviter ce type de situation est de limiter la croissance de l'arbre. On peut déclarer un nœud comme nœud terminal s'il ne parvient pas à être supérieur à un certain critère de taille ; c'est à dire, si $\#(\tau) \leq n_{min}$, où $\#(\tau)$ est le nombre d'échantillons dans le nœud τ et n_{min} est une certaine taille minimale déjà déclarée pour un nœud. Comme un nœud terminal ne peut pas être divisé en deux nœuds successeurs, il agit comme un frein à la croissance des arbres ; plus grande est la valeur de n_min, plus sévère est le frein. Une autre action est d'arrêter la division d'un nœud si la qualité de la division est inférieure à une certaine limite prédéterminée. Cependant, ces règles d'arrêt ne garantissent pas la sélection des meilleures branches des arbres.

Une meilleure approche a été développée par Breiman [16], l'approche repose sur le principe de faire pousser l'arbre jusqu'à saturation puis d'élaguer les branches (de bas en haut) jusqu'à ce que l'arbre ait la «bonne taille». Un arbre élagué est un sous-arbre du grand arbre original. La méthode d'élagage d'un arbre est la partie cruciale du processus. Il existe de nombreuses façons d'élaguer un grand arbre : nous décidons qui est le «meilleur» des sous-arbres en utilisant une estimation de pourcentage d'erreurs pour la classification $R(T)$, où T est l'arbre de classification.

L'estimation du pourcentage d'erreur pour la classification pour l'arbre T est donnée par l'équation 3.42, où $\widetilde{T} = \{\tau_1, \tau_2, ..., \tau_L\}$ dénote l'ensemble de tous les nœuds de l'arbre T et $P(\tau)$ est la probabilité qu'un échantillon tombe dans le nœud τ. On estime $P(\tau_l)$ par la proportion $p(\tau_l)$ de tous les échantillons qui tombent dans le nœud τ_l, l'estimation du pourcentage d'erreur donc est redéfinie par l'équation 3.43, où $R'(\tau_l) = r(\tau) p(\tau_l)$ et $r(\tau)$ est l'estimation du pourcentage d'erreur d'un échantillon dans le nœud τ qui est défini par l'équation 3.44 pour k classes et par l'équation 3.45 pour deux classes.

$$R(T) = \sum_{\tau \in \widetilde{T}} R(\tau) P(\tau) = \sum_{l=1}^{L} R(\tau_l) P(\tau_l) \tag{3.42}$$

$$R'(T) = \sum_{l=1}^{L} r(\tau_l) p(\tau_l) = \sum_{l=1}^{L} R'(\tau_l) \tag{3.43}$$

$$r(\tau) = 1 - \max_k p(k|\tau) \tag{3.44}$$

$$r(\tau) = 1 - \max(p, 1-p) = \min(p, 1-p) \tag{3.45}$$

L'algorithme d'élagage utilisé est le suivant :

1. On fait pousser un grand arbre jusqu'à T_{max}, où chaque nœud doit contenir au moins n_{min} échantillons.
2. On calcule l'estimation de $R(\tau)$ pour chaque nœud $\tau \in T_{max}$.
3. On élague T_{max} à la hausse vers le nœud racine, afin qu'à chaque étape l'estimation de $R(T)$ soit minimisée.

Afin de définir une approche de régularisation dans l'algorithme d'élagage, on définit un paramètre de complexité $\alpha \geq 0$ pour n'importe quel nœud $\tau \in T$, il est définit comme :

$$R_\alpha(\tau) = R'(\tau) + \alpha \tag{3.46}$$

Pour l'équation 3.46, la mesure du coût de la complexité de l'élagage pour un arbre est définie comme :

$$R_\alpha(T) = \sum_{l=1}^{L} R_\alpha(\tau_l) = R'(T) + \alpha|\widetilde{T}| \tag{3.47}$$

où $|\widetilde{T}| = L$ est le nombre de nœuds terminaux dans le sous-arbre T de T_{max}. Le terme $\alpha|\widetilde{T}|$ est un terme de pénalité pour la taille des arbres de sorte que $R_\alpha(T)$ pénalise $R'(T)$ pour générer un arbre trop grand. Pour chaque α, on choisit le sous-arbre $T(\alpha)$ de T_{max} que minimise $R_\alpha(T)$:

$$R_\alpha(T(\alpha)) = \min_T R\alpha(T) \tag{3.48}$$

Si le sous-arbre $T(\alpha)$ satisfait l'équation 3.48, il est donc appelé un sous-arbre réduit (ou un sous-arbre élagué de façon optimale) de l'arbre T_{max}. Pour tout α, il peut y avoir plus d'un sous arbre réduit de l'arbre T_{max}.

La valeur de α détermine la taille de l'arbre. Lorsque α est très petit, le terme de pénalité sera faible, et donc la taille du sous-arbre réduit $T(\alpha)$, qui sera essentiellement déterminé par $R'(T(\alpha))$, sera grande. Si $\alpha = 0$ et la croissance de l'arbre T_{max} assez grande de sorte que chaque nœud terminal contienne seulement un échantillon, alors chaque nœud terminal prend la classe de son échantillon solitaire, chaque échantillon est classé correctement et $R'(T_{max}) = 0$. Ainsi, T_{max} minimise $R_0(T)$. Ensuite, si α augmente la taille du sous-arbre $T(\alpha)$ sera réduit de plus en plus et il aura de moins en moins de nœuds terminaux. Lorsque α est très grand, on a élagué tout l'arbre T_{max} et il ne reste plus que le nœud racine.

Enfin, comment peut-on associer une classe avec un nœud terminal ? Supposons le nœud terminal τ qui a $n(\tau) = \#(\tau)$ échantillons, duquel $n_k(\tau) = \#(\tau|k)$ appartient à la classe Π_k, $k = 1, 2, ..., K$. Donc, la classe qui a le plus grand nombre d'échantillons est attribuée au nœud terminal τ. C'est ce qu'on appelle la règle de la pluralité. Cette règle peut être déduite de la loi de Bayes, où on attribue le nœud τ à la classe Π_i si $p(i|\tau) = \max_k p(k|\tau)$; si on estime la probabilité a priori π_k par $n_k(\tau)/n(\tau)$, $k = 1, 2, ..., K$, alors cela revient à la règle de la pluralité.

3.2.6 Evaluation globale de la méthode

Nous avons fait une première évaluation globale de notre approche de détection d'obstacles basée sur l'apparence [81, 37] en prenant initialement 60 images pour construire une base d'images dont 30 appartiennent à la classe d'intérêt (le sol) et 30 à différents obstacles. Chaque image de la base d'images a une taille de 100×100 pixels, donc la base de données initiale est composée de $600,000$ échantillons. 20% des échantillons de la base de données sont utilisés dans la phase d'apprentissage, les 80% restant étant utilisés dans la phase d'évaluation. Afin que cette dernière soit plus représentative et moins dépendante des échantillons sélectionnés, nous avons fait 20 séances d'entraînement dans les mêmes conditions, sauf pour les échantillons utilisés pendant la phase d'apprentissage qui sont choisis au hasard pour chaque entraînement. La performance de la méthode est donc évaluée en analysant statiquement les 20 séances d'entraînement de ces 80% d'échantillons classés par la méthode et en comptabilisant le nombre des échantillons bien triés.

Nous utilisons huit mesures d'évaluation pour comparer la performance de différents algorithmes de classification et être en mesure de régler les différents paramètres de notre méthode de détection d'obstacles. D'abord, nous définissons les mesures d'évaluation, soit $\left(x_1^t, y_1^t\right), \left(x_2^t, y_2^t\right), ..., \left(x_m^t, y_m^t\right)$, l'ensemble d'échantillons d'évaluation, où chaque x_i^t appartient à un domaine X dans l'espace $\mathbb{R}^r$, et

chaque étiquette y_i^t est un élément de l'ensemble Y. La méthode de classification estime donc l'échantillon x_i^t comme appartenant à la classe $\widetilde{y}_i^t$. L'échantillon est considéré comme positif s'il appartient à la classe d'intérêt (c'est à dire le sol) et négatif autrement. Les deux premières mesures, bien connues, sont les taux de reconnaissance et d'erreur (voir les équations 3.49 et 3.50) qui mesurent le nombre des échantillons bien et mal classés, respectivement). Le taux de reconnaissance ou d'exactitude n'est pas une mesure suffisante pour évaluer une méthode de classification. Il y a des cas où l'estimation d'un taux de reconnaissance peut induire une erreur sur la qualité d'un classificateur ou peut ne pas contenir suffisamment d'information pour l'évaluer [122]. Or, dans ces cas, les mesures de la sensibilité et la spécificité peuvent être utilisées comme une alternative ou un complément au taux d'exactitude [55].

$$\text{Taux de reconnaissance} = \frac{\text{Nombre d'échantillons bien classés}}{\text{Nombres d'échantillons}} = \frac{\#\left\{y^t | y_i^t = \widetilde{y}_i^t\right\}}{m} \tag{3.49}$$

$$\text{Taux d'erreur} = \frac{\text{Nombre d'échantillons mal classés}}{\text{Nombres d'échantillons}} = \frac{\#\left\{y^t | y_i^t \neq \widetilde{y}_i^t\right\}}{\#\left\{y^t\right\}} = \frac{\#\left\{y^t | y_i^t \neq \widetilde{y}_i^t\right\}}{m} \tag{3.50}$$

La mesure de sensibilité (également connue comme taux de vrais positifs ou «recall») évalue dans quelle mesure le classificateur peut reconnaître les échantillons positifs et la sensibilité est définie comme :

$$\text{Sensibilité} = \frac{\text{Nombre d'échantillons positifs bien classés}}{\text{Nombre d'échantillons positifs}} = \frac{\#\left\{y^t | y_i^t = 1 \& y_i^t = \widetilde{y}_i^t\right\}}{\#\left\{y^t | y_i^t = 1\right\}} \tag{3.51}$$

La spécificité (également connue comme taux de vrais négatifs) mesure à quel point le classificateur peut reconnaître les échantillons négatifs. La spécificité est définie comme :

$$\text{Spécificité} = \frac{\text{Nombre d'échantillons négatifs bien classés}}{\text{Nombre d'échantillons négatifs}} = \frac{\#\left\{y^t | y_i^t = -1 \& y_i^t = \widetilde{y}_i^t\right\}}{\#\left\{y^t | y_i^t = -1\right\}} \tag{3.52}$$

Les quatre autres mesures complémentaires de la performance sont la précision, le taux de fausses alarmes, les vraisemblances positives et négatives. La précision évalue combien d'échantillons classés comme appartenant à la classe «positive» sont en effet «positifs», formellement la précision est définie comme :

$$\text{Précision} = \frac{\text{Nombre d'échantillons positifs bien classés}}{\text{Nombre d'échantillons classés positifs}} = \frac{\#\left\{y^t | y_i^t = 1 \& y_i^t = \widetilde{y}_i^t\right\}}{\#\left\{y^t | \widetilde{y}_i^t = 1\right\}} \tag{3.53}$$

Le taux de fausses alarmes ou taux de faux positifs mesure à quel point le classificateur peut omettre de reconnaître les échantillons négatifs. Le taux de fausses alarmes est défini comme :

$$\text{Taux de fausses alarmes} = \frac{\text{Nombre d'échantillons négatifs mal classés}}{\text{Nombre d'échantillons négatifs}} = \frac{\#\left\{y^t | y_i^t = -1 \& y_i^t \neq \widetilde{y}_i^t\right\}}{\#\left\{y^t | y_i^t = -1\right\}} \tag{3.54}$$

La vraisemblance positive évalue la proportion d'échantillons positifs bien classés par les échantillons négatifs mal classés. Cette mesure est pondérée par le nombre d'échantillons dans chaque classe. La vraisemblance positive est définie comme :

$$\text{Vraisemblance positive} = \frac{\text{Sensibilité}}{1 - \text{Spécificité}} = \frac{\#\left\{y^t | y_i^t = -1\right\}}{\#\left\{y^t | y_i^t = 1\right\}} \times \frac{\#\left\{y^t | y_i^t = 1 \& y_i^t = \widetilde{y}_i^t\right\}}{\#\left\{y^t | y_i^t = -1 \& y_i^t \neq \widetilde{y}_i^t\right\}} \quad (3.55)$$

La vraisemblance négative mesure la proportion d'échantillons classés comme négatifs ; cette mesure est pondérée par le nombre d'échantillons dans chaque classe. La vraisemblance négative est définie comme :

$$\text{Vraisemblance négative} = \frac{1 - \text{Sensibilité}}{\text{Spécificité}} = \frac{\#\left\{y^t | y_i^t = -1\right\}}{\#\left\{y^t | y_i^t = 1\right\}} \times \frac{\#\left\{y^t | y_i^t = 1 \& y_i^t \neq \widetilde{y}_i^t\right\}}{\#\left\{y^t | y_i^t = -1 \& y_i^t = \widetilde{y}_i^t\right\}} \quad (3.56)$$

La première évaluation a été la comparaison entre les trois algorithmes de classification. Cette évaluation nous permet de trouver l'algorithme de classification qui a la meilleure performance. La table 3.4 montre les moyennes des résultats obtenus pour cette évaluation. Nous évaluons trois méthodes de classification : l'algorithme k-PPV [109], l'algorithme SVM [89, 124] avec trois noyaux : linéaire, quadratique et cubique et l'algorithme AdaBoost. Nous avons utilisé les mêmes échantillons pour chaque séance d'entraînement pour les trois méthodes de classification. Chaque échantillon est composé de 42 caractéristiques ou attributs, 2 attributs pour la couleur et 40 pour la texture. Les attributs de texture correspondent aux attributs de la moyenne, le contraste, l'homogénéité, la variance et la corrélation pour 2 distances de 1 et 2 pixels et pour 4 directions de $0°$, $45°$, $90°$ et $135°$ dans une fenêtre de traitement de 17×15 pixels.

TABLE 3.4 – Table comparative des huit mesures de la performance pour les trois algorithmes de classification.

	k-PPV	SVM			AdaBoost
		Linéaire	Quadratique	Cubique	
Taux de reconnaissance	0.9365	0.9222	0.9501	0.9367	0.9676
Taux d'erreur	0.0635	0.0778	0.0499	0.0633	0.0324
Sensibilité	0.9019	0.9050	0.9355	0.9148	0.9725
Spécificité	0.9712	0.9395	0.9647	0.9586	0.9627
Précision	0.9690	0.9375	0.9637	0.9568	0.9631
Taux de fausses alarmes	0.0288	0.0605	0.0353	0.0414	0.0373
Vraisemblance positive	31.271	15.264	28.176	22.803	26.083
Vraisemblance négative	0.1011	0.1011	0.0669	0.0888	0.0286

Dans la table 3.4, on peut souligner que l'algorithme AdaBoost a les meilleures performances pour le taux de reconnaissance, la sensibilité et la vraisemblance négative. L'algorithme k-PPV a les meilleures performances pour la spécificité, la précision, le taux de fausses alarmes et la vraisemblance positive. Cependant, la performance évaluée pour ces quatre mesures n'est pas très éloignée de la performance donnée par l'algorithme AdaBoost. Inversement la performance de l'algorithme k-PPV mesurée par les taux de reconnaissance, sensibilité et vraisemblance négative est beaucoup plus faible, ce qui revient à dire que l'algorithme AdaBoost est globalement le meilleur classificateur. Il est également le meilleur algorithme pour faire la détection d'obstacles et le deuxième meilleur pour faire la détection du sol. En revanche, l'algorithme k-PPV est le meilleur pour faire la détection du sol, mais sa performance en détection d'obstacles est très faible, ce qui affecte son rendement global et en fait l'avant dernier

algorithme de classification en fonction de la performance globale. L'algorithme SVM à l'égard de sa performance est moyen, sans exceller dans aucun cas. Toutefois, on peut insister sur le fait que le noyau quadratique a la meilleure performance par rapport aux autres deux noyaux. Sur la base de cette analyse nous avons choisi l'algorithme AdaBoost comme classificateur de base pour la méthode de détection d'obstacles basée sur l'apparence.

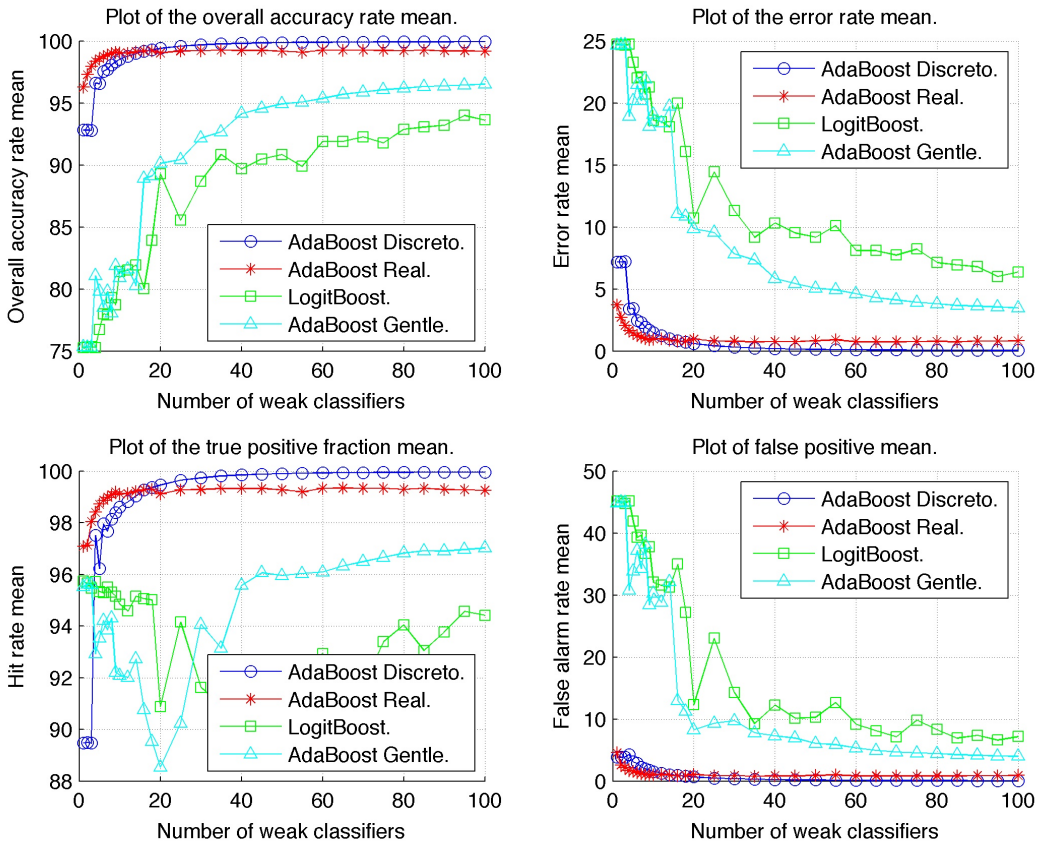
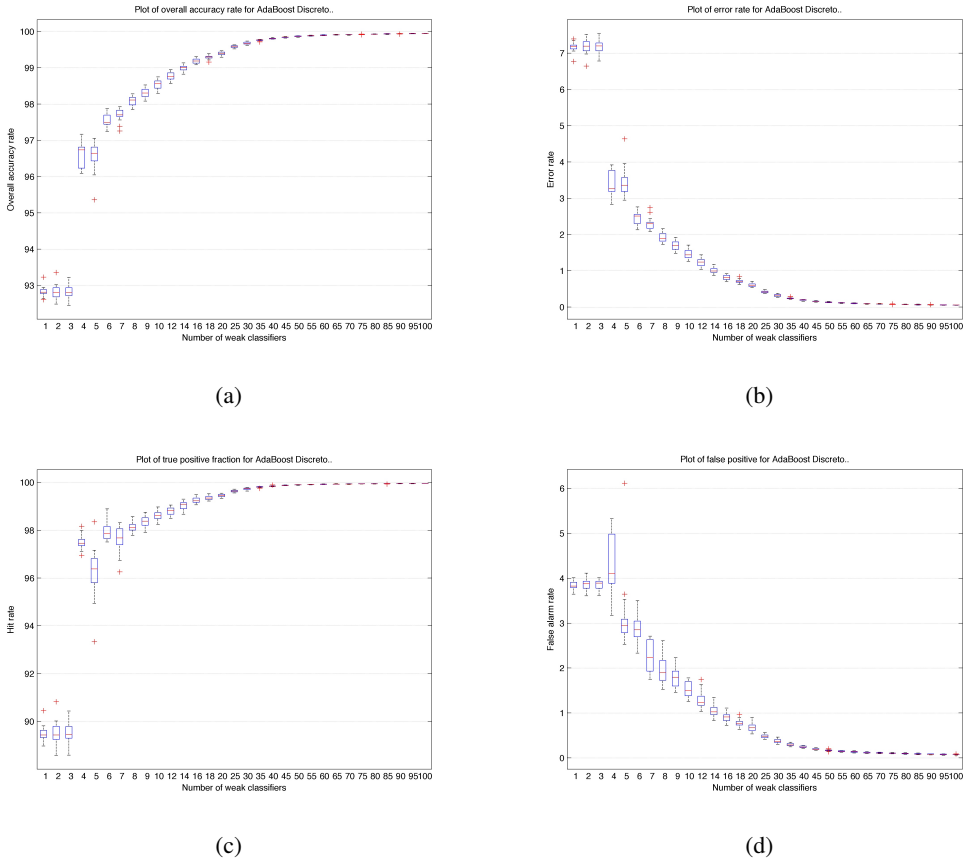


FIGURE 3.11 – Graphiques de la performance pour différents nombres d'hypnothèses faibles dans la gamme de 1 à 100 et les 4 types d'entraînement de l'algorithme AdaBoost, (a) taux de reconnaissance, (b) taux d'erreur, (c) précision et (d) taux de fausses alarmes.

L'étape suivante dans notre évaluation est de trouver la méthode la plus appropriée pour faire l'apprentissage dans l'algorithme boosting. Pour cette raison nous avons testé les quatre méthodes les plus couramment utilisées : les algorithmes AdaBoost discret [60], AdaBoost réel [60], LogitBoost [46] et AdaBoost Gentle [46]. Pour ces différents algorithmes l'évaluation de la performance sera réduite à seulement quatre mesures lesquelles donnent suffisamment d'information pour établir un comparatif et faire un choix. Nous allons utiliser le taux de reconnaissance et d'erreur pour mesurer la performance globale, la précision pour évaluer la performance pour la détection du sol et le taux de fausses alarmes pour mesurer la performance pour la détection d'obstacles. Pour cette évaluation, nous avons utilisé les mêmes paramètres que précédemment pour calculer les échantillons. La figure 3.11 montre les quatre graphiques des moyennes des mesures pour comparer les performances entre les quatre méthodes d'apprentissage. On peut observer que les performances données par les algorithmes LogitBoost et AdaBoost Gentle sont les plus faibles et les plus instables ; au contraire les performances données par les

algorithmes AdaBoost discret et AdaBoost réel sont les meilleures et ont une plus grande stabilité en fonction du nombre d'hypothèses faibles. Il est important de préciser que pour une gamme de 1 à 18 hypothèses faibles l'algorithme AdaBoost réel a la meilleure performance, mais pour une gamme entre 19 et 100 hypothèses faibles la meilleure performance est donnée par l'algorithme AdaBoost discret, lequel correspond à la performance souhaitée. Pour cette raison, nous avons retenu ce dernier pour faire l'apprentissage du classificateur.



(a) (b)

(c) (d)

FIGURE 3.12 – Graphiques à moustaches de la performance pour différents nombres d'hypothèses faibles dans la gamme de 1 à 100 pour l'algorithme AdaBoost discret, (a) taux de reconnaissance, (b) taux d'erreur, (c) précision et (d) taux de fausses alarmes.

L'évaluation qui suit a pour but de mesurer la performance en fonction du nombre d'hypothèses faibles pour l'algorithme AdaBoost discret. Nous avons utilisé les mêmes paramètres pour calculer les attributs que dans les deux dernières évaluations. La figure 3.12 montre les quatre graphiques à moustaches correspondant aux quatre mesures pour évaluer la performance. Les graphiques à moustaches nous permettent de faire une analyse plus détaillée de l'information statistique et nous pouvons ainsi observer le comportement de la performance en fonction du nombre d'hypothèses faibles. Ces graphiques nous permettent de souligner un point d'intérêt qui se trouve dans la position correspondant à un nombre de 20 hypothèses faibles : ce point correspond au point d'inflexion où la performance est stabilisée et donne le meilleur compromis.

Les trois évaluations précédentes nous ont permis de trouver l'algorithme de classification le plus approprié à notre problème de détection et nous ont également permis de régler les paramètres de cet algorithme. Les évaluations suivantes nous permettent de régler les paramètres dans le calcul de la texture,

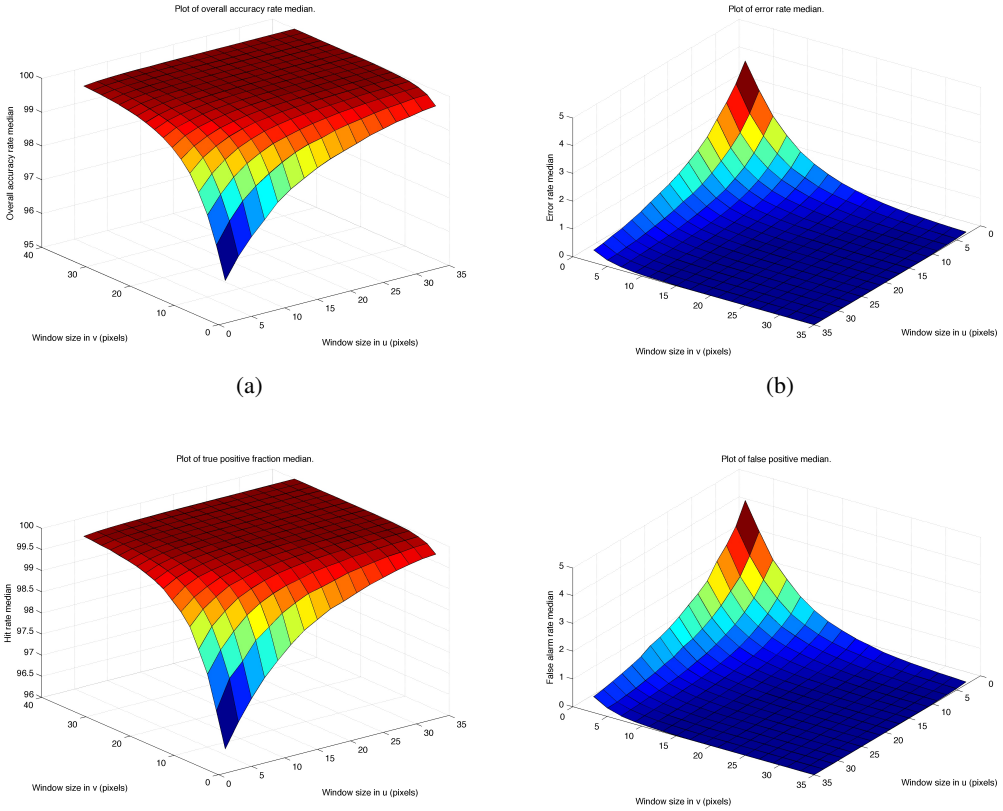


FIGURE 3.13 – Graphiques de la performance médiane pour différentes tailles de la fenêtre de traitement dans l'espace des attributs de texture entre 3 et 35 pixels aux coordonnées u et v, en utilisant 20 hypothèses faibles pour l'entraînement de l'algorithme AdaBoost discret et 42 attributs, (a) taux de reconnaissance, (b) taux d'erreur, (c) précision et (d) taux de fausses alarmes.

ce qui réduira le temps de calcul, les ressources nécessaires et optimisera les performances lors de la classification. Le premier paramètre à analyser est la taille de la fenêtre de traitement. La figure 3.13 montre les graphiques des médianes de mesures des performances ; de façon empirique nous avions pensé que la performance fournie par les deux coordonnées u et v (horizontale et verticale, respectivement) dans la fenêtre de traitement était la même. En d'autres termes la performance donnée par un classificateur avec une taille de fenêtre de traitement de 15×17 est la même qu'un autre classificateur avec une taille de 17×15. Cependant on peut constater, à partir des graphiques que ce n'est pas vrai, vu que la coordonnée v influe plus directement sur la performance que la coordonnée u. C'est pourquoi, il est préférable de choisir la coordonnée v plus grande que u. A partir de cette hypothèse, nous avons choisi une taille de la fenêtre de traitement de 15×17 car Elle correspond au point d'inflexion dans la performance souhaitée.

La prochaine étape dans notre évaluation est de choisir le nombre et la valeur des distances pour les déplacements afin de calculer les images des sommes et des différences. Nous avons effectué des tests avec quatre distances différentes allant de 1 à 4 pixels. Pour cette évaluation, nous nous sommes appuyés sur les résultats des évaluations précédentes. Il est donc évalué l'algorithme AdaBoost discret avec 20 hypothèses faibles et une fenêtre de traitement de 15×17 pixels pour calculer les attributs de texture. Lors de la dernière étape de notre évaluation, nous chercherons quelles sont les directions les plus significatives pour effectuer ces déplacements ; à ce stade, nous utilisons encore les quatre directions.

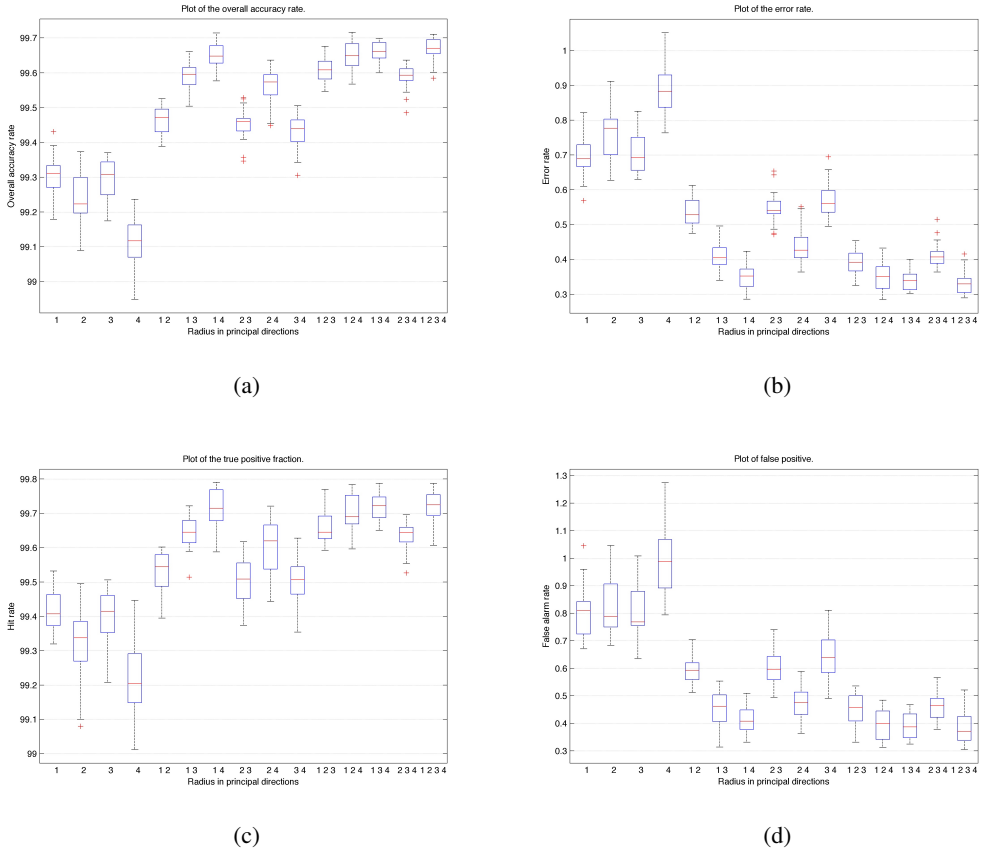


FIGURE 3.14 – Graphiques à moustaches de la performance pour différentes distances dans l'espace des attributs de texture entre 1 et 4 pixels, en utilisant 20 hypothèses faibles pour l'entraînement de l'algorithme AdaBoost discret avec une fenêtre de traitement de 15×17 pixels et 4 directions, (a) taux de reconnaissance, (b) taux d'erreur, (c) précision et (d) taux de fausses alarmes.

De manière empirique, on peut supposer que la richesse d'un modèle est directement proportionnelle au nombre des attributs et donc plus on a d'informations sur celui-ci meilleure sera la classification. Toutefois, il est bien connu qu'il y a certains attributs dans l'espace des caractéristiques qui ne peuvent pas fournir d'informations complémentaires ou fournir des informations qui existent déjà dans le modèle. Pour cette raison, il est important de choisir les paramètres sur la base des contraintes particulières qui donnent l'information la plus pertinente pour notre problème de classification. Ceci aura également pour effet, pour un niveau de performances souhaitées, de réduire le temps de calcul et les ressources nécessaires.

La figure 3.14 montre les graphiques à moustaches des mesures de la performance. On peut voir que la performance pour les quatre distances individuelles est moins bonne que pour toutes les combinaisons entre elles. Jusque là l'hypothèse empirique est vraie mais lorsqu'on combine deux distances cette hypothèse n'est plus valide. On peut noter que la performance médiane donnée par la combinaison des distances de 1 et 4 pixels est supérieure à la performance médiane obtenue par deux des quatre combinaisons de trois distances et pas très éloignée pour les deux autres combinaisons. De plus, cette même performance obtenue pour la combinaison de 1 et 4 pixels est très proche de la performance obtenue pour la combinaison des quatre distances. Ceci signifie que l'information contenue dans le modèle est plus riche entre plus éloignées soient les distances entre eux et non par le nombre de distances. En conclusion

la combinaison la plus appropriée pour notre méthode de détection est la combinaison des deux distances de 1 et 4 pixels.

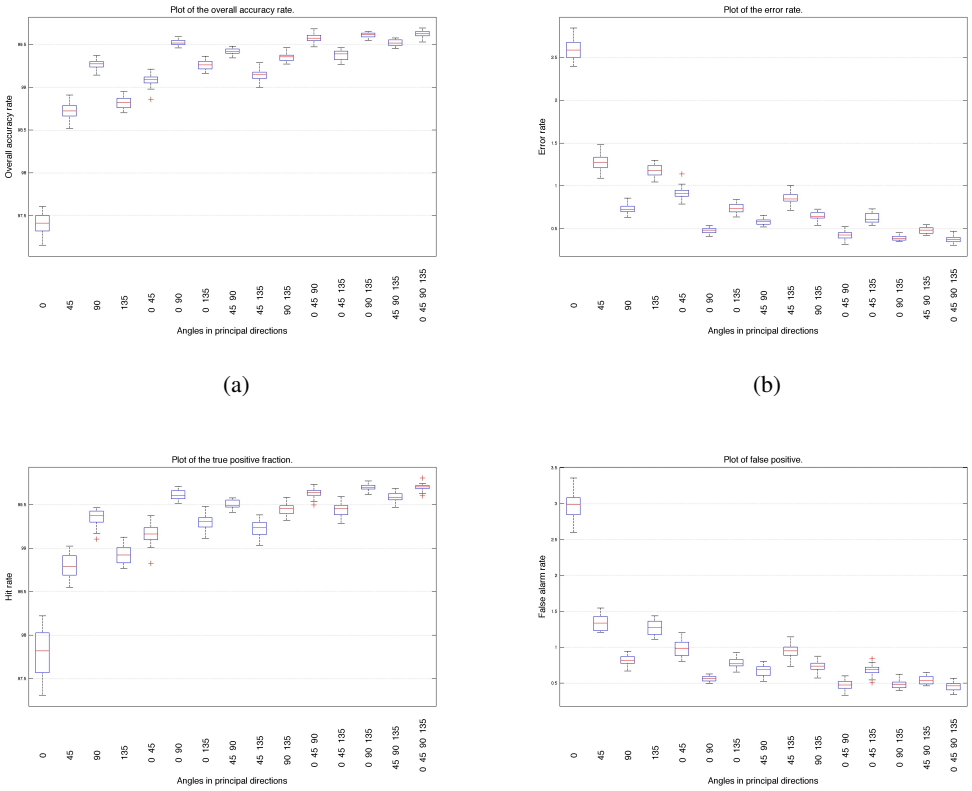


FIGURE 3.15 – Graphiques à moustaches de la performance pour différentes directions dans l'espace des attributs de texture pour 0°, 45°, 90° et 135°, en utilisant 20 hypothèses faibles pour l'entraînement de l'algorithme AdaBoost discret avec une fenêtre de traitement de 15×17 pixels et 2 directions de 1 et 4 pixels, (a) taux de reconnaissance, (b) taux d'erreur, (c) précision et (d) taux de fausses alarmes.

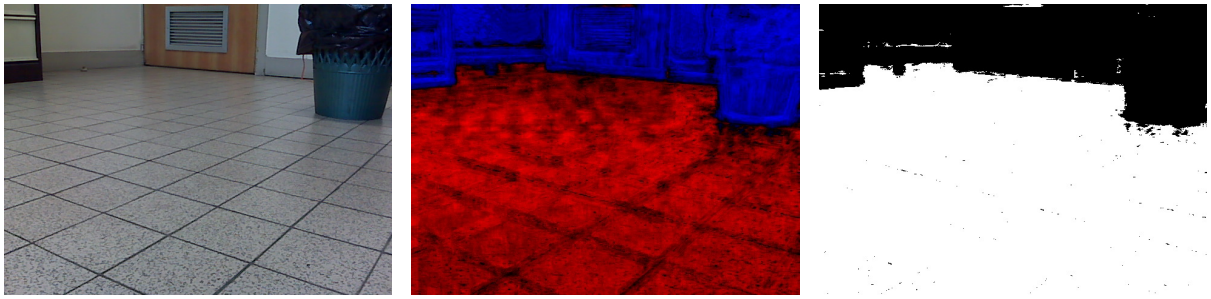
La dernière évaluation est relative aux directions dans les déplacements utilisées pour calculer les images de sommes et différences. Ces directions sont : 0°, 45°, 90° et 135°. la figure 3.15 montre les graphiques à moustaches des mesures pour évaluer la performance. On peut voir que la meilleure performance est obtenue pour la direction individuelle de 90°. Il est important de souligner que la médiane de la performance pour la combinaison des directions de 0° et 90° est supérieure aux autres combinaisons de deux directions et aussi supérieure à deux des quatre combinaisons des trois directions. De plus cette médiane est très proche des deux autres combinaisons et de la combinaison des quatre directions.

3.2.7 Résultats expérimentaux

Les images testées ont été acquises avec la caméra numérique STMicroelectronics IMG-724-E01 [130]. La détection d'obstacles a été effectuée en utilisant les résultats des évaluations réalisées dans la section précédente. La méthode de classification utilisée est donc l'algorithme AdaBoost discret avec 20 hypothèses faibles. La modélisation du sol a été faite en utilisant 22 attributs de la couleur et de la texture, 2 pour la couleur et 20 pour la texture. Les composantes chromatiques a et b dans l'espace de couleur CIE-Lab ont été utilisées comme des attributs de couleur, tandis que la composante de luminosité L^* a

été utilisée pour obtenir les attributs de la texture. Les attributs de texture ont été calculés en utilisant les 5 caractéristiques de texture définies par la méthode de l'adéquation des histogrammes de sommes et différences. Ces caractéristiques de texture ont été calculées à partir des images de sommes et différences, lesquelles ont été obtenues pour 4 déplacements. Les déplacements utilisés pour obtenir les images des sommes et différences correspondent à 2 distances de 1 et 4 pixels et 2 directions de $0°$ et $90°$.

La méthode finale pour la détection d'obstacles devant être mise en œuvre sur une architecture embarquée, il est nécessaire que celle-ci soit structurée autant que possible. Ainsi, nous avons choisi de faire une détection d'obstacles dense, c'est à dire que le traitement est fait pixel par pixel. Les attributs sont calculés dans une fenêtre de traitement qui est déplacé dans l'image et les régions définies par la fenêtre de traitement présentent des chevauchements.



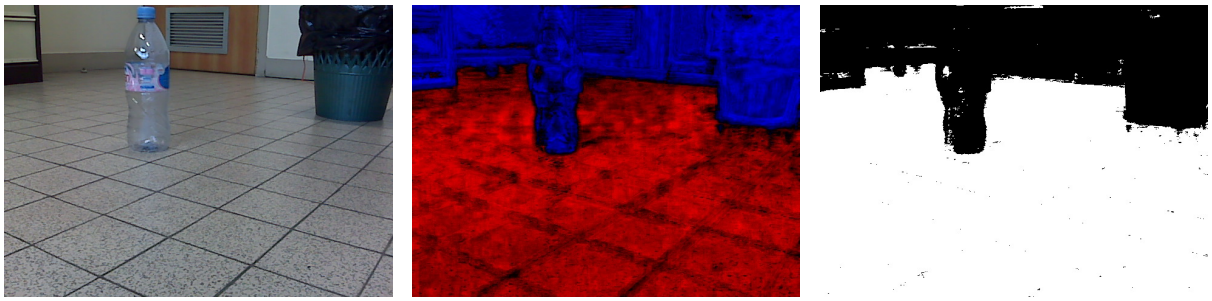
(a) (b) (c)

FIGURE 3.16 – Détection d'obstacles basée sur des attributs de couleur et de texture pour une scène presque sans obstacles, (a) l'image originale, (b) l'image de probabilité de détection d'obstacles et (c) l'image binaire de détection d'obstacles

La figure 3.16 présente la détection d'obstacles dans une scène d'intérieur comportant peu d'obstacles. La figure 3.16a est l'image originale dans l'espace de couleur RGB. La figure 3.16b montre l'image de probabilité résultant de la méthode de classification. Dans cette image les pixels rouges représentent une probabilité négative d'appartenance à la classe d'intérêt, c'est à dire qu'ils n'appartiennent pas au sol, et donc ils ont une forte probabilité d'appartenir à la classe obstacles. En revanche, les pixels bleus représentent une probabilité positive d'appartenance à la classe sol. Les pixels noirs ont une probabilité proche de zéro et ne peuvent donc pas être classés en toute sécurité. La figure 3.16c montre le résultat de l'application d'un seuil dans l'image de probabilité afin d'obtenir une image binaire, dans laquelle il est plus facile de déterminer si le pixel appartient au sol ou non. Dans notre méthode nous avons utilisé la fonction signe, c'est à dire nous avons évalué si la probabilité est positive ou négative. On peut noter que les résultats de la détection d'obstacles s'avèrent corrects.

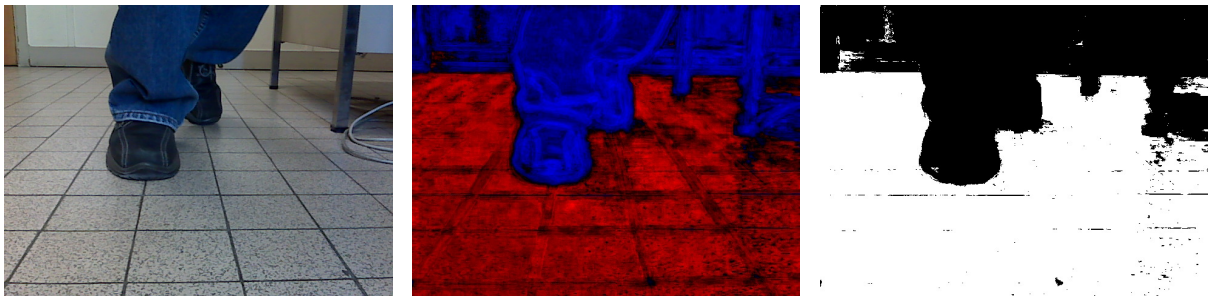
La figure 3.17 présente la détection d'une bouteille vide en plastique vue comme un obstacle. Ce cas particulier est intéressant parce que la bouteille reflète la couleur et la texture du sol. Ce type d'objet est difficile à détecter à partir d'une technique basée seulement sur la couleur ou la texture, contrairement à une technique hybride comme la notre, laquelle résout le problème dans une certaine mesure du fait que la réflexion n'est pas parfaite.

La figure 3.18 illustre deux situations intéressantes dans la détection des obstacles : la première est la détection d'un piéton, ce qui est bien résolu par notre méthode ; la seconde est la détection du sol avec des changements d'éclairage ou d'ombres : cette situation est relativement bien évaluée du fait de l'immunité au changement de l'illumination dans l'espace de couleur. Cependant, on peut noter que lorsqu'il y a un grand changement d'éclairage notre méthode ne peut pas bien classer les régions. Dans la figure 3.18c, on peut observer qu'en dessous du bureau, il y a une forte ombre portée sur le sol. Cette région n'est pas classée correctement. Néanmoins, l'ombre moins intense autour de ce bureau et l'ombre du piéton sont



(a) (b) (c)

FIGURE 3.17 – Détection d'obstacles basée sur des attributs de couleur et de texture pour une scène avec une bouteille considérée comme un obstacle pour la méthode, (a) l'image originale, (b) l'image de probabilité de détection d'obstacles et (c) l'image binaire de détection d'obstacles



(a) (b) (c)

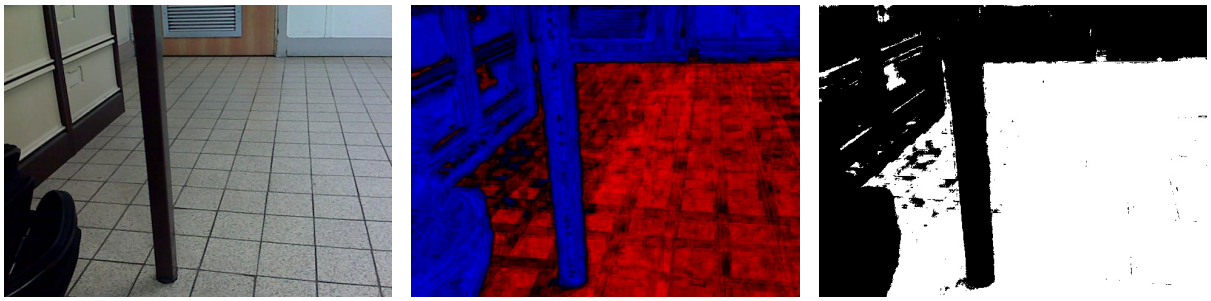
FIGURE 3.18 – Détection d'obstacles basée sur des attributs de couleur et de texture pour une scène comportant un piéton considéré comme un obstacle pour la méthode, (a) l'image originale, (b) l'image de probabilité de détection d'obstacles et (c) l'image binaire de détection d'obstacles

bien classés.

La figure 3.19 montre la dernière scène analysée dans cette section. Nous pouvons y noter deux situations intéressantes pour la détection d'obstacles. La première est la détection d'obstacles depuis un point de vue totalement différent de celui utilisé pendant la phase d'apprentissage. On peut voir, malgré cette perspective différente que le sol a été bien classé. La seconde est la classification des obstacles inconnus qui ne sont pas dans la base d'apprentissage utilisée lors de l'entraînement. Dans cette scène on trouve le sac à dos et un meuble à clapets situé dans l'image à gauche. Ces objets sont bien classés bien que l'on note l'existence de petites zones de l'étagère qui sont classés comme du sol. Cela est dû au fait que certains attributs de la couleur et de la texture peuvent être semblables à ceux du sol. En conséquence ils peuvent être classés de manière incorrecte. Notons toutefois que la performance globale de la classification pour ces situations est acceptable pour notre application et pour la plupart des applications.

3.3 Détection d'obstacles pour une approche géométrique

La détection d'obstacles basée sur l'apparence présente un inconvénient inhérent à la méthode. En effet, dans cette approche un obstacle est défini comme un objet qui n'a pas les mêmes attributs de couleur et de texture que le sol. Par exemple une feuille de papier posée par terre sera considérée comme



(a) (b) (c)

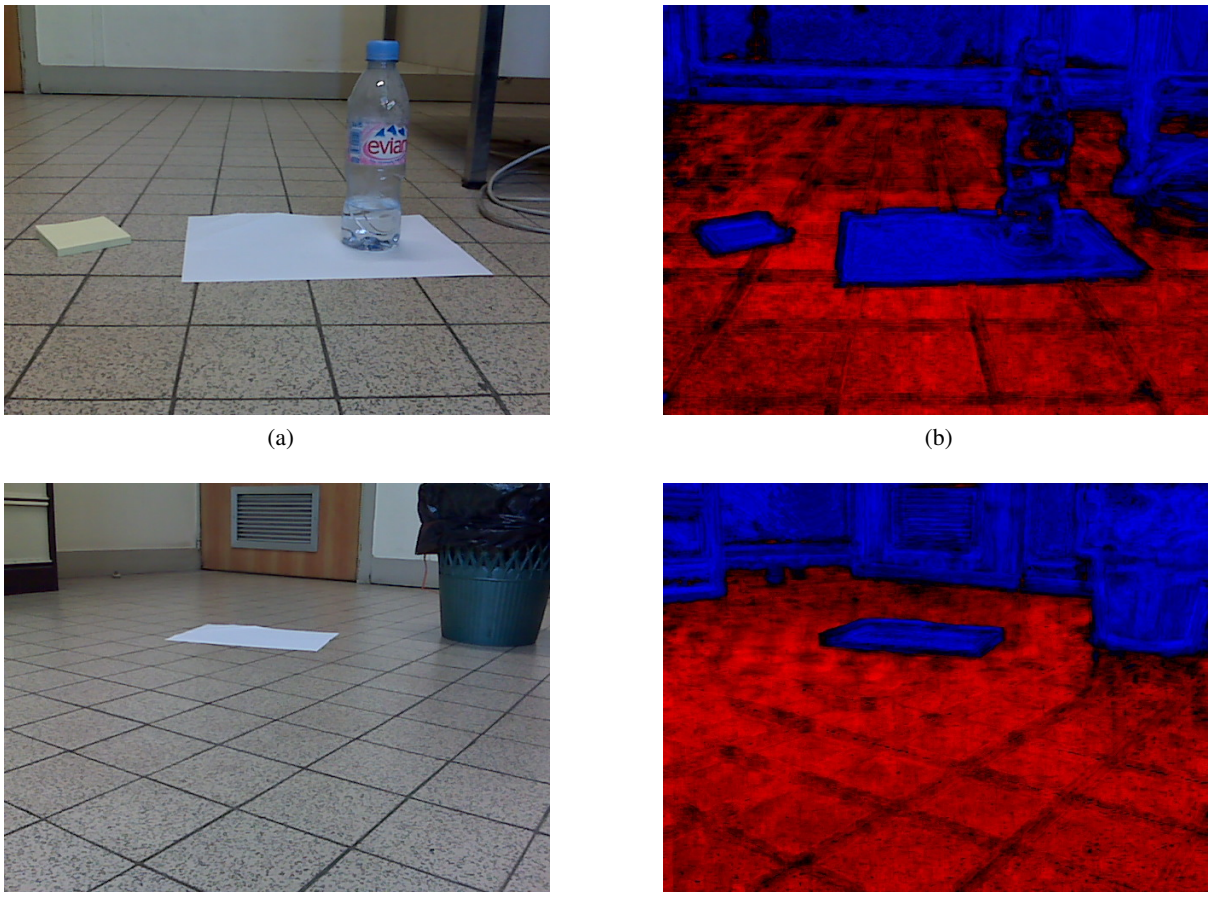
FIGURE 3.19 – Détection d'obstacles basée sur des attributs de couleur et de texture pour une scène acquise avec un angle de vue différent, (a) l'image originale, (b) l'image de probabilité de détection d'obstacles et (c) l'image binaire de détection d'obstacles

un obstacle. Il est donc nécessaire de définir un obstacle plus strictement, en considérant notamment que tout changement dans l'élévation du plan du sol constitue un danger pour la navigation du robot. La feuille de papier devient donc une fausse détection parce qu'elle ne représente pas un danger. Deux exemples de ces fausses détections sont illustrés sur la figure 3.20. Les figures 3.20a et 3.20c présentent les images des scènes originales qui seront traitées pour obtenir les images de probabilité de détection (voir figures 3.20b et 3.20d), dans lesquelles l'intensité de couleur bleue représente la grandeur de la probabilité de détection d'un obstacle et l'intensité de couleur rouge représente la grandeur de la probabilité de détection de la classe sol. Dans les figures 3.20b et 3.20d, on peut voir que la bouteille, la feuille de papier, les post-it, la poubelle, le mur et le bureau ont été détectés comme des obstacles, mais les post-it et la feuille de papier ne représentent pas un danger en soi. Si on souhaite donc optimiser la trajectoire du robot ces fausses détections pourraient la modifier. Cet inconvénient n'est cependant pas très grave car la méthode garantit la détection du sol et donc la sécurité du robot.

Afin de réduire le taux de fausses détections donné par la première approche, nous faisons appel à une deuxième méthode complémentaire de détection d'obstacles. Cette méthode utilise le mouvement du robot ou l'ego-motion lors de sa navigation dans un environnement intérieur. Ce mouvement dû à l'ego-motion peut être obtenu grâce à une centrale inertielle (Inertial Mesurements Units IMU) montée sur le robot ou estimé en utilisant directement les images [91].

Nous avons choisi d'estimer le mouvement en utilisant la centrale inertielle de notre robot d'expérimentation car en plus d'avoir des mesures de déplacement du robot synchrones avec la prise d'images, l'implémentation matérielle d'une telle solution est largement plus simple que l'exploitation des images elles-mêmes. Pour faire cette estimation nous considérerons que la caméra est montée de manière fixe sur le robot et donc que son déplacement est le même. En conséquence le modèle que nous avons établi considère uniquement les degrés de liberté du robot, ce qui rend l'estimation de l'ego-motion exacte et plus stable.

Dans cette section nous décrirons la méthode de projection des points depuis le plan image au monde environnant, en faisant la projection au sol, puis le processus inverse à l'image suivante, c'est à dire, les points projetés sur le sol dans le monde sont transformés en utilisant les données d'odométrie récupérées depuis la centrale inertielle. Ces points sont comparés à ceux des points dans l'image prise dans l'instant suivant : les points qui sont au niveau du sol seront comparés à eux-mêmes, de sorte que ces points auront une magnitude proche de zéro et différente si le point est au-dessus ou au-dessous du sol. Dans la suite nous présenterons l'information essentielle sur les transformations géométriques de la caméra et la cinématique des corps rigides puis nous présenterons la transformation de perspective inversée qui est la méthode de détection d'obstacles.



(a) (b)

(c) (d)

FIGURE 3.20 – Détection d'obstacles basée sur les attributs de couleur et de texture pour différents objets considérés comme obstacles pour la méthode, (a) et (c) images originales, (b) et (d) images de probabilité de détection d'obstacles

3.3.1 Transformations géométriques de la caméra

Le processus de formation d'une image dans une caméra comporte deux étapes :

1. D'abord, une projection des rayons de lumière incidents à partir de l'objectif de la caméra, vers les points photosensibles constituant le capteur d'image.
2. Une conversion de chaque point photosensible en un signal électrique qui une fois numérisé fournit l'image digitale.

Du point de vue de la perspective, un rayon est le trajet d'un point dans le monde vers le centre de la caméra. Cette procédure est appelée projection centrale. Ensuite chaque rayon est propagé par le système optique jusqu'à l'intersection avec le plan image où il forme une image du point [44]. Cette tâche suppose une projection idéale avec une distorsion optique minimale voire nulle. La figure 3.21 montre le modèle sténopé, très répandu dans la littérature de vision par ordinateur, du principe de la projection centrale. Ce modèle considère que la longueur focale f est équivalente à la distance entre le centre optique C et le plan image, le centre optique se trouvant dans le plan focal, lequel est lui-même perpendiculaire à l'axe optique et parallèle au plan image. Le point d'intersection de l'axe optique avec le plan image est appelé point principal. Dans le cas où la longueur focale du modèle sténopé est l'unité, i.e. $f = 1$ alors

la projection sténopée est considérée comme *normalisée* et on obtiendra une image *normalisée*.

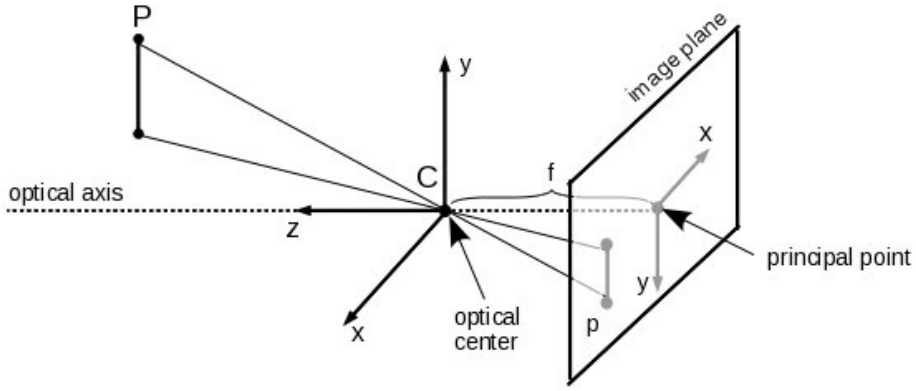


FIGURE 3.21 – Modèle de projection sténopé de la caméra.

Deux transformations nous permettent d'associer la position d'un point de l'espace 3D aux cordonnées pixel de sa projection dans l'image :

1. La projection du point 3D sur le plan image 2D.
2. La transformation affine qui change les coordonnées d'un point du plan image du repère métrique de la camera en un repère pixel lié à l'image.

La première transformation est définie par les triangles semblables sur le schéma 3.21, duquel il découle que le point sur le plan image $\mathbf{p} = (x, y)^T$ d'un point 3D $\mathbf{P} = (X, Y, Z)^T$, par rapport au système de coordonnées de la caméra, est donné par :

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{f}{Z} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.57}$$

En coordonnées homogènes, la projection perspective du point *P* en utilisant les vecteurs homogènes $p = (x, y, 1)^T$ et $P = (X, Y, Z, 1)^T$ obtenue à partir des cordonnées Euclidiennes en les augmentant de 1, la projection de l'équation 3.57 peut se réécrire par l'équation matricielle suivante :

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \sim s \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \underbrace{\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{M}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.58}$$

où $\mathbf{M}$ est la matrice de projection dans l'espace $\mathbb{R}^{3\times4}$, habituellement, cette matrice est décomposée en :

$$M = M_f M_0 = \underbrace{\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{M_f}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{M_0}} \tag{3.59}$$

où la matrice $\mathbf{M_0}$ est la matrice de projection normalisée. Cette matrice permet la projection du plan image à longueur focale de $z = 1$. La matrice $\mathbf{M_f}$ contient le facteur d'échelle pour avoir le plan

image à une distance $z = f$ différente de l'unité. L'équation 3.58 de transformation du point 3D $P = (X, Y, Z, 1)^T$ dans le point $p = (x, y, 1)^T$ sur le plan image peut se réécrite :

$$s\mathbf{p} = M_f M_0 \mathbf{P} \tag{3.60}$$

En outre, le modèle sténopé est une projection idéale qui ne considère pas les erreurs dans la formation d'une image depuis un objectif réel, spécialement quand celui-ci est de qualité médiocre. La figure 3.22 montre une image présentant une distorsion couramment observée, notamment sur les zones éloignées du centre : il s'agit d'une distorsion radiale [59].

FIGURE 3.22 – Déformation radiale lors de la formation de l'image dans la caméra, principalement sur les bords de l'image.

Les coordonnées déformées de l'image $\mathbf{p_d} = (x_d, y_d)^T$ exprimées en fonction des coordonnées de l'image $\mathbf{p} = (x, y)^T$ sont représentées par l'équation 3.61 laquelle utilise un modèle simple avec coefficients de distorsion k_i.

$$\mathbf{p}_d = \left(1 + k_1 \parallel \mathbf{p} \parallel^2 + k_2 \parallel \mathbf{p} \parallel^4\right) \mathbf{p} \tag{3.61}$$

Les pixels de l'image délivrée par la caméra sont numérotés à partir du coin supérieur gauche. De plus ces pixels ne sont pas forcement carrés et parfois non-orthogonaux, par conséquent il est nécessaire d'introduire des notions d'échelle et de compensation. Ces aspects sont pris en compte pour formuler un modèle de transformation affine afin de convertir les coordonnées métriques de l'image déformée $\mathbf{p_d} = (x_d, y_d, 1)^T$ en coordonnées du pixel $\mathbf{p_i} = (u, v, 1)^T$,

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{M_s}} \begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix} \tag{3.62}$$

où u_0 et v_0 sont les coordonnées en pixels du point principal. Notons que s_u et s_v sont respectivement les facteurs d'échelle vertical (pixels par mètre) et horizontal et le paramètre d'obliquité ou "*skew*" s_θ lequel est souvent proche de zéro. Soit la distorsion négligeable, alors $\mathbf{p_d} = \mathbf{p}$ et l'équation 3.62 est une transformation affine représentant un changement d'échelle, une rotation et une translation. La relation entre les coordonnées $(X, Y, Z)^T$ du point $\mathbf{P}$ et les coordonnées image (u, v) du point $\mathbf{p_i}$ est donnée par :

$$u = s_u f \frac{X}{Z} + u_0 \qquad v = s_v f \frac{Y}{Z} + v_0 \tag{3.63}$$

En multipliant l'équation de transformation affine 3.62 avec celle de la projection perspective 3.58 nous pouvons obtenir la matrice des paramètres intrinsèques $\mathbf{K}$:

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{K}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{3.64}$$

où les paramètres $\alpha_u = s_u f$, $\alpha_v = s_v f$, u_0 et v_0 sont les paramètres intrinsèques de la caméra et $\mathbf{K}$ représente la matrice de calibration.

Une calibration préalable de la caméra sera nécessaire pour estimer les paramètres intrinsèques ; ces paramètres sont nécessaires pour calculer le rayon optique associé à un quelconque pixel de l'image.

Avec ce modèle fondamental de la caméra, il est possible de trouver une ligne, à partir des coordonnées de l'image, dans laquelle se trouve le point 3D correspondant. Cependant, du fait qu'une information est perdue lors de la transformation, la notion de profondeur associée à ce point est perdue. Le modèle inverse de projection perspective de la caméra est utilisé pour transformer le point $\mathbf{p_i} = (u, v)^T$ dans l'image dans son correspondant 3D $\mathbf{P} = (X, Y, Z)^T$. Cependant, cette transformation n'ayant pas le facteur d'échelle correspondant à la dimension perdue, la projection du point 3D est faite sur le plan image normalisé, puis le point est projeté vers une profondeur connue ou estimée. Le modèle inverse est réalisé en deux étapes, en supposant que la distorsion soit négligeable :

1. La projection inverse du pixel est la transformation du point $\mathbf{p_i} = (u, v, 1)^T$ en coordonnées de l'image (pixels) dans le point $\mathbf{p} = (x, y, 1)^T$ en coordonnées métriques du plan image normalisé. Cette transformation est définie par l'inverse de l'équation 3.64 comme :

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} 1/\alpha_u & -\alpha_\theta/\alpha_u\alpha_v & (\alpha_\theta v_0 - \alpha_v u_0)/\alpha_u\alpha_v \\ 0 & 1/\alpha_v & -v_0/\alpha_v \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K^{-1}}} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{3.65}$$

2. La projection inverse du point depuis le plan image normalisé est utilisée pour transformer le point $\mathbf{p} = (x, y, 1)^T$ dans le plan image avec $f = 1$ en un point 3D $\mathbf{P} = (X, Y, Z)^T$. Cette projection est définie pour l'équation de la droite passant par le point du centre optique et le point (x, y) sur le plan image (voir figure 3.21) et peut être exprimée par la forme paramétrique suivante :

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = s\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.66}$$

où s représente la "*profondeur*" du point 3D $\mathbf{P} = (X, Y, Z)^T$ par rapport au plan focal, cette distance est calculée par l'équation de droite avec connaissance a priori ou estimation de la profondeur.

Notons par ailleurs que notre connaissance a priori de la profondeur est réduite aux points qui se trouvent sur le sol, sachant que tous les objets qui sont en-dessous ou au-dessus du niveau du sol sont considérés comme des obstacles.

3.3.2 Mouvement de corps rigides

Nous nous sommes intéressés à l'analyse du mouvement des corps rigides sans prendre en compte la masse ou la force de l'objet c'est à dire uniquement sa cinématique [62]. Des axes de coordonnées sont

assignés au corps rigide afin d'en décrire ses mouvements dans le temps. Ces mouvements comprennent deux transformations fondamentales : la translation et la rotation qui seront mesurées avec la centrale inertielle. La figure 3.23 montre les trois axes définis pour le robot X_R, Y_R, Z_R et les angles de rotation $\Delta\varphi, \Delta\alpha, \Delta\Psi$ (*roll, pitch, yaw*) qui y sont respectivement attachés. Ainsi le système de coordonnées créé peut être déplacé et orienté dans la direction des trois axes ce qui donne les six degrés de liberté du robot.

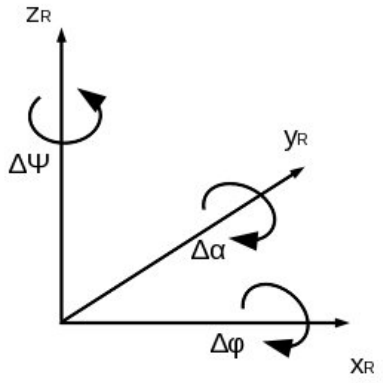


FIGURE 3.23 – Système de coordonnées du robot avec six degrés de liberté.

Le mouvement de translation du robot est positif en direction des axes XYZ de la figure 3.23. Par contre les angles sont considérés positifs en accord avec les conventions suivantes :

- $\Delta\varphi$ est positive si le côté droit du robot va vers le bas et le côté gauche se lève autour de l'axe X.
- $\Delta\alpha$ est positive si le devant du robot, autour de l'axe Y, pointe ver le bas.
- $\Delta\Psi$ est positive si le robot tourne à gauche, autour de l'axe Z.

La position et l'orientation d'un objet par rapport au repère d'un autre objet (par exemple le robot par rapport au monde) peuvent être décrites par le couple $\{\mathbf{d}, \mathbf{R}\}$, où $\mathbf{d}$ décrit la distance de l'objet par rapport à l'origine du repère, appelé *vecteur de translation* et $\mathbf{R}$ décrit l'orientation de l'objet par rapport au repère, appelée *matrice de rotation*.

Le vecteur de translation est composé de trois distances : une pour chaque axe du repère. Ce vecteur est défini par l'équation suivante :

$$\mathbf{d} = \begin{pmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{pmatrix} \tag{3.67}$$

La matrice de rotation peut être représentée de différentes façons. Nous utilisons deux représentations pour travailler avec l'orientation : les *angles d'Euler* et le *quaternion*.

Les angles d'Euler sont les trois angles de rotation qui représentent une orientation dans l'espace 3D. Cette orientation est obtenue depuis les trois rotations consécutives autour des trois différents axes. Nous utilisons la convention ZYX correspondant à l'ordre dans lequel les axes sont mis en rotation. Cette convention prend en compte l'axe Z (l'angle yaw $\Delta\Psi$) comme le premier axe à mettre en rotation, puis, la deuxième rotation autour de l'axe Y (l'angle pitch $\Delta\alpha$) déjà mis en rotation, enfin, la troisième rotation autour de l'axe X (l'angle roll $\Delta\varphi$) deux fois déjà mis en rotation.

Soit $\mathbf{e} = (\Delta\varphi, \Delta\alpha, \Delta\Psi)^T$ les angles d'Euler correspondant aux angles roll, pitch et yaw, la matrice de rotation est obtenue à partir de l'équation suivante :

$$\mathbf{R}(\mathbf{e}) = \begin{bmatrix} \cos\Delta\alpha\cos\Delta\Psi & \sin\Delta\varphi\sin\Delta\alpha\cos\Delta\Psi - \cos\Delta\varphi\sin\Delta\Psi & \cos\Delta\varphi\sin\Delta\alpha\cos\Delta\Psi + \sin\Delta\varphi\sin\Delta\Psi \\ \cos\Delta\alpha\sin\Delta\Psi & \sin\Delta\varphi\sin\Delta\alpha\sin\Delta\Psi + \cos\Delta\varphi\cos\Delta\Psi & \cos\Delta\varphi\sin\Delta\alpha\sin\Delta\Psi - \sin\Delta\varphi\cos\Delta\Psi \\ -\sin\Delta\alpha & \sin\Delta\varphi\cos\Delta\alpha & \cos\Delta\varphi\cos\Delta\alpha \end{bmatrix} \tag{3.68}$$

Les angles d'Euler sont une représentation minimale pour les rotations. Ils ont l'inconvénient de présenter des discontinuités ce qui nécessite de prendre en compte le domaine de validité pour chacun d'eux :

$$\Delta\varphi \in [-\pi, \pi], \qquad \Delta\alpha \in [pi/2, \pi/2], \qquad \Delta\Psi \in [0, 2\pi]. \tag{3.69}$$

La deuxième représentation pour la matrice de rotation que nous avons utilisée est basée sur les quaternions. Les quaternions, notés $\mathbb{H}$, sont un système de nombres qui étend le champ des nombres complexes, extension similaire à celle qui avait conduit les nombres réels $\mathbb{R}$ à celui des nombres complexes $\mathbb{C}$. Un quaternion est un nombre hypercomplexe.

Soit Z_1 et Z_2 deux nombres complexes sous forme algébrique $Z_1 = a + bi \in \mathbb{C}$, $Z_2 = c + di \in \mathbb{C}$; $\forall a, b, c, d \in \mathbb{R}$, où i représente l'unité imaginaire et telle que $i^2 = -1$. Nous définissons le quaternion $Q \in \mathbb{H}$ comme :

$$Q = Z_1 + Z_2 j \in \mathbb{H}; \qquad \forall Z_1, Z_2 \in \mathbb{C} \tag{3.70}$$

où j est un nombre particulier tel que $i^2 = j^2 == k^2 = ijk = -1$ et $ij = -ji = k$. Tout quaternion $Q \in \mathbb{H}$ peut être considéré comme une combinaison linéaire des quatre quaternions unités 1, i, j et k, alors l'équation 3.70 peut également s'écrire $Q = a + bi + cj + dk$ ou sous la forme vectorielle comme :

$$\mathbf{q} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \in \mathbb{R}^4 \tag{3.71}$$

Le conjugué du quaternion $\mathbf{q} = (a, b, c, d)^T$ est le quaternion obtenu en conservant la partie réelle et en prenant les opposés des parties imaginaires $\mathbf{q}^* = \bar{\mathbf{q}} = (a, -b, -c, -d)^T$. Le produit du quaternion $\mathbf{q} = (a, b, c, d)^T$ par son conjugué $\mathbf{q}^*$ donne $\mathbf{q} \cdot \mathbf{q}^* = a^2 + b^2 + c^2 + d^2$ qui est un nombre réel positif. La norme du quaternion $\mathbf{q} = (a, b, c, d)^T$ est le nombre réel positif défini comme $||\mathbf{q}|| = \sqrt{\mathbf{q} \cdot \mathbf{q}^*}$. Si $||\mathbf{q}|| = 1$ le quaternion est appelé *normalisé*.

Les nombres complexes normalisés peuvent être utilisés pour représenter toute rotation dans le plan Euclidien. Les quaternions peuvent donc être utilisés de la même façon dans l'espace Euclidien 3D. Soit q un quaternion normalisé qui sera toujours représenté par :

$$\mathbf{q} = \begin{pmatrix} \cos\theta \\ u_x \sin\theta \\ u_y \sin\theta \\ u_z \sin\theta \end{pmatrix} \tag{3.72}$$

où $(u_x, u_y, u_z)^T = \mathbf{u}^T$ est un vecteur unitaire $||u|| = 1$ et θ est un scalaire. Un vecteur $\mathbf{v} \in \mathbb{R}^3$ dans l'espace du quaternion $\mathbb{H}$ est défini comme $\mathbf{q_v} = 0 + v_x i + v_y j + v_z k \in \mathbb{H}$, une rotation de θ radians autour de l'axe défini par le vecteur unitaire $\mathbf{u}$ pour le vecteur $\mathbf{v}$ est défini comme :

$$\mathbf{q_w} = \mathbf{q} \cdot \mathbf{q_v} \cdot \mathbf{q}^* \tag{3.73}$$

Cette expression est une forme matricielle équivalent à $\mathbf{w} = \mathbf{R}\mathbf{v}$ laquelle définit la matrice de rotation $\mathbf{R}$ comme une fonction du quaternion $\mathbf{q} = (a, b, c, d)^T$ avec :

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} a^2+b^2+c^2+d^2 & 2\cdot(bc-ad) & 2\cdot(bd+ac) \\ 2\cdot(bc+ad) & a^2-b^2+c^2-d^2 & 2\cdot(cd-ab) \\ 2\cdot(bd-ac) & 2\cdot(cd+ab) & a^2-b^2-c^2+d^2 \end{bmatrix} \tag{3.74}$$

La rotation inverse est obtenue à partir du quaternion conjugué $\mathbf{q}^*$, donc $\mathbf{R}(\mathbf{q}^*) = \mathbf{R}^T(\mathbf{q})$. Le quaternion opposé représente exactement la même rotation que celui d'origine, c'est à dire $\mathbf{R}(-\mathbf{q}) = \mathbf{R}(\mathbf{q})$. La composition des rotations satisfait la relation $\mathbf{R}(\mathbf{q_1}\cdot\mathbf{q_2}) = \mathbf{R}(\mathbf{q_1})\mathbf{R}(\mathbf{q_2})$. Cela montre que le quaternion n'est pas une représentation minimale pour les rotations, mais que la contrainte de normalisation doit être impérativement assurée après toute modification.

La figure 3.24 illustre la transformation linéaire 3D du repère robot au repère monde avec un vecteur de translation $\mathbf{d}$ et une matrice de rotation $\mathbf{R}$ obtenue à partir des angles d'Euler ou des quaternions. Cette transformation linéaire de repère est faite en utilisant des coordonnées homogènes. Le point $\mathbf{P} \in \mathbb{R}^3$ dans la représentation homogène est défini par l'équation 3.75 et les transformations linéaires du repère robot au repère monde et son inverse sont décrites par les équations 3.76 et 3.77 respectivement. Les équations 3.78 et 3.79 illustrent la notation de façon compacte pour les transformations linéaires homogènes.

$$\widehat{\mathbf{P}} = \begin{pmatrix} \mathbf{P} \\ 1 \end{pmatrix} \in \mathbb{R}^4 \tag{3.75}$$

$$\begin{pmatrix} \mathbf{P}^M \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} \mathbf{R} & \mathbf{d} \\ \mathbf{0} & 1 \end{bmatrix}}_{\mathbf{H}^{MR}} \begin{pmatrix} \mathbf{P}^R \\ 1 \end{pmatrix} \tag{3.76}$$

$$\begin{pmatrix} \mathbf{P}^R \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{d} \\ \mathbf{0} & 1 \end{bmatrix}}_{\mathbf{H}^{RM}} \begin{pmatrix} \mathbf{P}^M \\ 1 \end{pmatrix} \tag{3.77}$$

$$\widehat{\mathbf{P}}^M = \mathbf{H}^{MR}\widehat{\mathbf{P}}^R \tag{3.78}$$

$$\widehat{\mathbf{P}}^R = \mathbf{H}^{RM}\widehat{\mathbf{P}}^M \tag{3.79}$$

La figure 3.25 illustre la transformation linéaire 3D du repère robot au repère du plan image (aussi connu comme repère frame) qui est obtenue par l'équation suivante :

$$\widehat{\mathbf{P}}^F = \mathbf{H}^{FC}\mathbf{H}^{CR}\widehat{\mathbf{P}}^R \tag{3.80}$$

où $\mathbf{H}^{FC}$ et $\mathbf{H}^{CR}$ représentent les matrices des paramètres extrinsèques. La matrice $\mathbf{H}^{CR}$ est obtenue à partir de la position (X_C, Y_C, Z_C) et de l'orientation (φ, α, Ψ) de la caméra sur le robot. La matrice $\mathbf{H}^{FC}$ est utilisée pour changer l'orientation du repère caméra au repère du plan image. Les deux matrices sont statiques, une fois qu'elles sont obtenues elles ne changent pas, réduisant ainsi le coût de calcul lors du traitement.

Une transformation linéaire 3D complète d'un point $\mathbf{P}^O$ qui appartient à l'objet O observé par la caméra C est représentée par l'équation 3.81. Le point $\mathbf{P}^F$ contient les coordonnées du point $\mathbf{P}^O$ par rapport au repère du plan image de la caméra C

$$\widehat{\mathbf{P}}^F = \mathbf{H}^{FC}\mathbf{H}^{CR}\mathbf{H}^{RM}\widehat{\mathbf{P}}^O \tag{3.81}$$

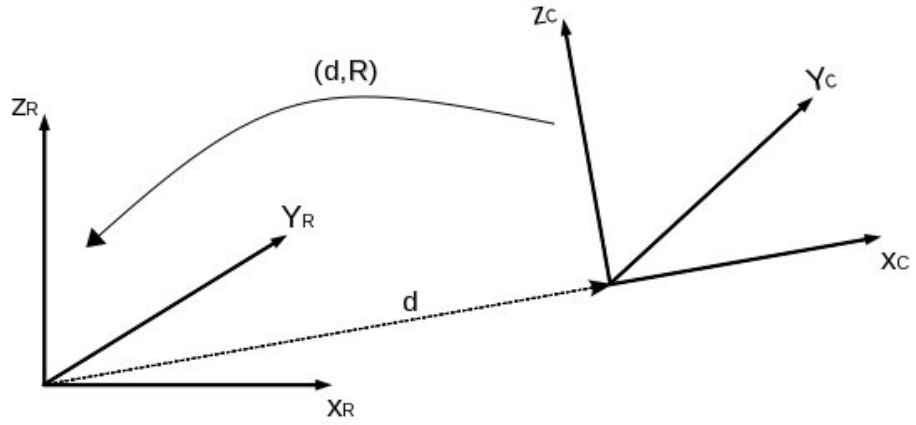


FIGURE 3.24 – Transformation 3D avec un vecteur de translation $\mathbf{d}$ et une matrice de rotation $\mathbf{R}$ du repère caméra au repère robot.

où $\mathbf{H}^{RM}$ représente le mouvement du robot par rapport au repère du monde. Le mouvement par rapport au système de coordonnées de la caméra dépendra donc de sa position dans le monde. La figure 3.25 représente les axes de référence utilisés pour la configuration de la scène pendant la navigation du robot. Le robot est un véhicule à 3 roues, non-holonome, et est donc limité à deux types de mouvements : une translation le long de l'axe X et une rotation autour de l'axe Z.

3.3.3 Transformation de perspective inversée

La perception d'une scène par la caméra dépend de l'angle de vue de celle-ci au moment d'acquérir l'image et de la distance entre la caméra et chaque objet de la scène. Étant donné que la caméra est montée dans une position et une orientation fixes sur le robot, l'angle de vue est donné par l'orientation et la position du robot dans la scène et par l'orientation et la position de la caméra par rapport au robot. La distance entre la caméra et chaque objet ne peut pas être récupérée directement à partir d'une seule image car il faut se rappeler de l'effet de la perspective au moment de l'acquisition de l'image. Cet effet est à prendre en compte lorsque les images seront traitées, en particulier si une approche géométrique est utilisée car ce facteur influe sur les informations contenues dans chaque pixel [12].

Pour faire face à ce problème, nous avons utilisé une approche géométrique. Cette approche consiste à projeter les pixels de l'image acquise dans le temps $(t-1)$ vers l'image acquise dans le temps (t), en supposant que les points 3D se situent au niveau du sol. Cette hypothèse est considérée du fait qu'on veut détecter les objets qui se trouvent en dessous ou au dessus du niveau du sol. Ainsi les pixels ne répondant pas à ces hypothèses seront considérés comme des obstacles. Soit $\widehat{\mathbf{p}}^i\left(t-1\right)=\left(u\left(t-1\right),v\left(t-1\right),1\right)^T$ les coordonnées de l'image acquise au temps $(t-1)$, alors la projection de ces coordonnées vers l'image acquise au temps (t) donnera les coordonnées estimées du pixel $\widetilde{\mathbf{p}}^i\left(t\right)=\left(\widetilde{u}\left(t\right),\widetilde{v}\left(t\right),1\right)^T$ calculées par l'équation suivante :

$$\widetilde{\mathbf{p}}^i\left(t\right)=\mathbf{K}\mathbf{H}^{FC}\mathbf{H}^{CR}\mathbf{H}^{ODO}\left(t-1,t\right)\mathbf{H}^{RC}\mathbf{H}^{CF}\mathbf{K}^{-1}\widehat{\mathbf{p}}^i\left(t-1\right) \tag{3.82}$$

où $\mathbf{K}$ et $\mathbf{K}^{-1}$ sont la matrice des paramètres intrinsèques et son inverse, respectivement. Les matrices $\mathbf{H}^{CF}$ et $\mathbf{H}^{FC}$ sont respectivement la matrice de transformation du repère du plan image au repère caméra

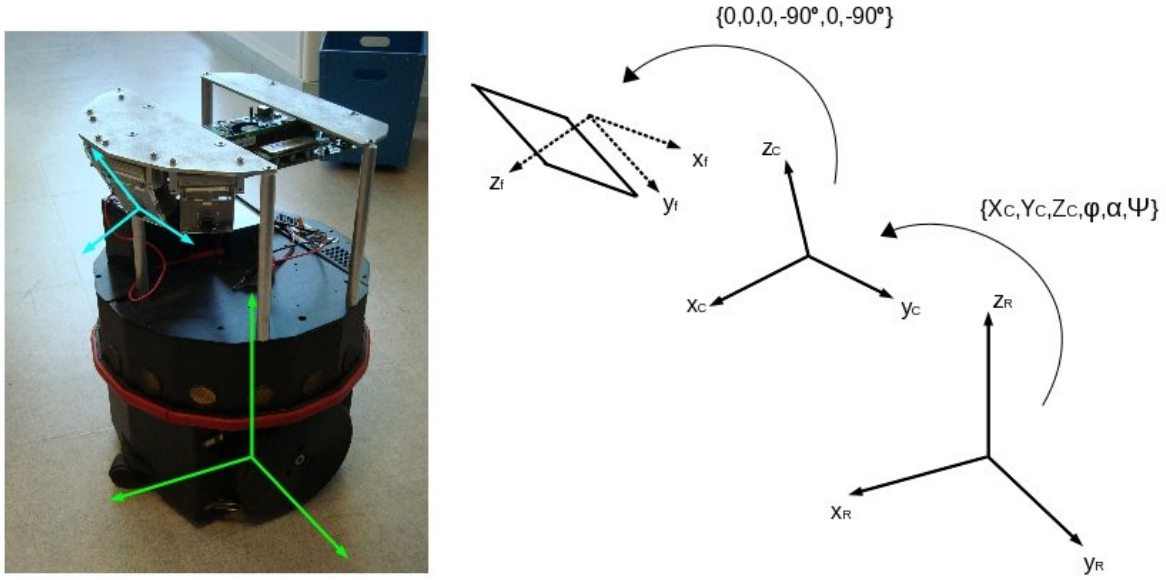


FIGURE 3.25 – Transformation 3D du repère robot au repère du plan image.

et son inverse,. Les matrices $\mathbf{H}^{RC}$ et $\mathbf{H}^{CR}$ sont respectivement la matrice de transformation du repère caméra au repère robot et son inverse. La matrice $\mathbf{H}^{ODO}\,(t-1,t)$ est la matrice de transformation du repère robot du temps $(t-1)$ au repère robot du temps t . Cette dernière matrice est obtenue par le déplacement du robot, lequel change à chaque acquisition d'image. Elle est la seule à être recalculée pour chaque acquisition puisque les paramètres intrinsèques et la position de la caméra par rapport au robot sont fixes.

Afin de réduire les temps de calcul nécessaires durant la navigation, nous utilisons l'hypothèse que le sol est plat ce qui limite les mouvements du robot à trois degrés de liberté. En conséquence le calcul pour obtenir la matrice $\mathbf{H}^{ODO}\,(t-1,t)$ se réduit au déplacement sur le plan *XY* et à une rotation sur l'axe *Z*.

La figure 3.26 présente la simulation de la transformation géométrique pour les quatre coins d'un rectangle situé au niveau du sol depuis deux positions différentes du robot. La figure de gauche montre la simulation de la scène en 3D : le rectangle sur le sol est en couleur marron, les deux positions du repère du robot et de la caméra sont en couleur bleue et rouge respectivement, les plans images correspondant sont en couleur jaune, les projections des quatre coins du rectangle sur les plans images sont en couleur cyan et finalement les champs de vue sont en couleur verte. La caméra est placée sur le robot dans la position $(X_C, Y_C, Z_C) = (0.5, 0, 0.4)$ avec un angle d'inclinaison de $\Delta\alpha = 20°$. Pour la première position simulée, le robot est à son origine au temps $(t-1)$ et nous utilisons U_{t-1} pour faire référence à cette position. Au temps (t), le robot se déplace à la position $U_t = (0.7, 1.2)$ sur le plan *XY* avec une orientation de $\Delta\Psi = -30°$. À droite dans la même figure, le premier graphique montre la projection du rectangle sur le sol vue depuis une caméra embarquée sur le robot à la position U_{t-1}. Les quatre cercles en couleur rouge représentent les quatre coins du rectangle. Le deuxième graphique montre la projection du rectangle vue par la caméra depuis la deuxième position U_t. Ici les quatre coins du rectangle sont représentés par les carrés en couleur bleue. Le troisième graphique montre, en rouge, la transformation géométrique du premier graphique vers la deuxième position U_t. On peut voir que si l'objet est sur le sol, les coordonnées des pixels issues de la transformation géométrique vont correspondre à celles de la projection du rectangle à la position U_t : il est donc possible de considérer ces pixels comme appartenant

au sol. Cela est démontré dans ce graphique car en gardant la projection du rectangle à la position U_t (graphique 2 en couleur bleue), il est possible de voir que la correspondance par rapport aux quatre coins du rectangle obtenus soit par transformation ou par projection est presque exacte. Le quatrième graphique a été obtenu à partir d'une transformation similaire à celle de la troisième sauf que la transformation a été faite dans le sens inverse, c'est-à-dire que l'image prise à la deuxième position U_t a été transformée vers l'image prise à U_{t-1}. Donc, si notre hypothèse est satisfaite les coordonnées des pixels sur le sol vont correspondre comme dans le cas du graphique 3.

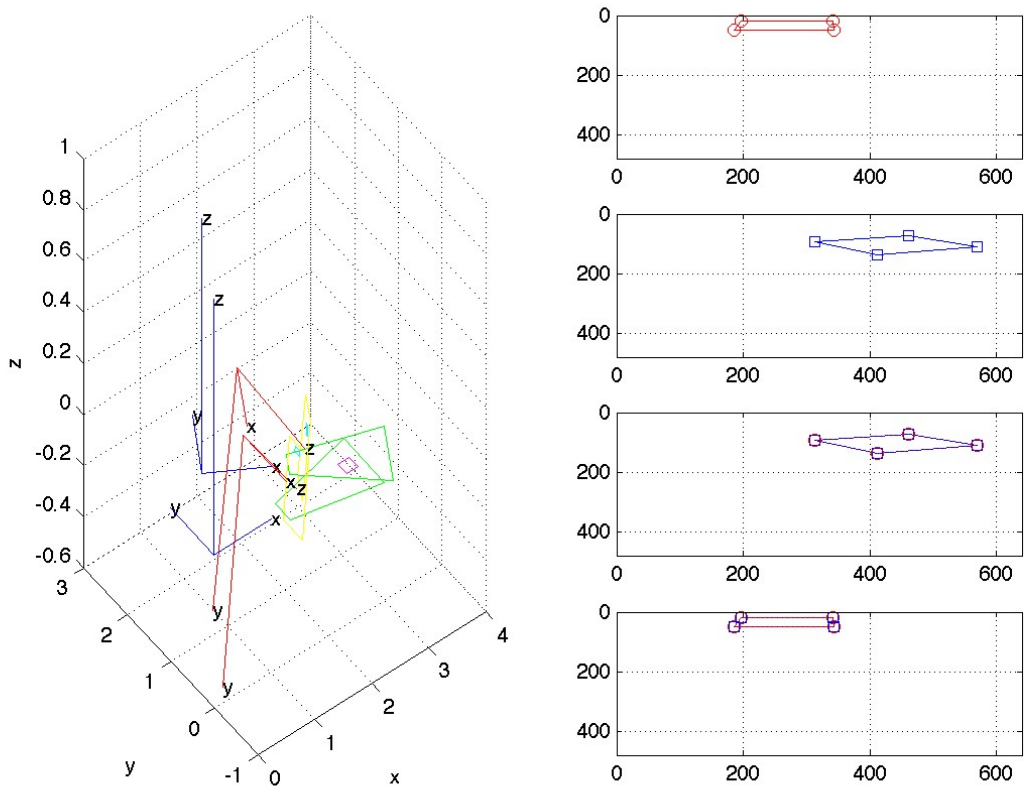


FIGURE 3.26 – Simulation de la transformation géométrique d'un rectangle au niveau du sol depuis deux points de vue différents de la caméra. À gauche la représentation 3D de la scène avec les deux positions du robot et la caméra. Les deux premiers graphiques à droite représentent les vues de la caméra dans la position au temps $(t-1)$ et (t). Le troisième graphique montre la projection de la première vue vers la deuxième (de $(t-1)$ vers (t)). Finalement le quatrième graphique montre la projection dans le sens inverse (de (t) vers $(t-1)$).

La figure 3.27 présente une simulation de la transformation géométrique similaire à la précédente, sauf que dans cette simulation, le rectangle est au-dessus du sol à une hauteur de 0.01 par rapport au plan XY qui représente le sol. On peut voir que dans ce cas les coordonnées des pixels entre l'image et sa transformation géométrique ne correspondent pas de sorte qu'il serait possible d'affecter ces pixels à un obstacle. Il est important de remarquer que les différences entre les coordonnées sont une fonction de la hauteur de l'obstacle, ce qui fait qu'il serait possible de le mesurer.

L'étape suivante dans notre méthode de détection des obstacles est la création d'une grille d'oc-

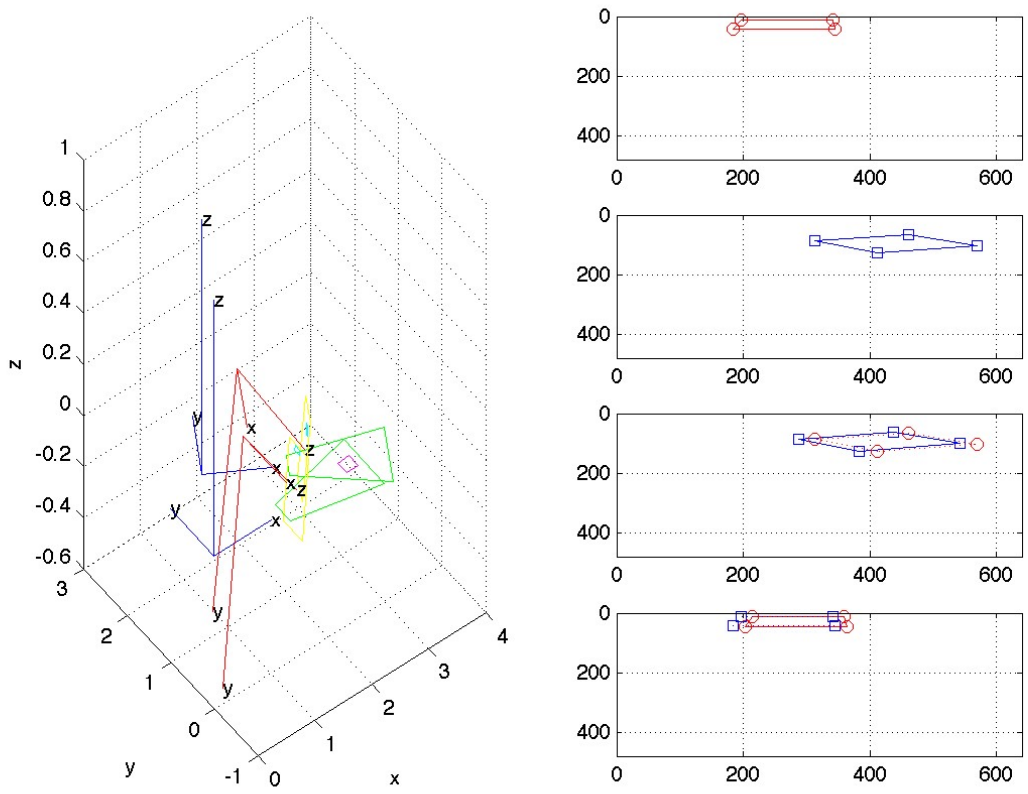


FIGURE 3.27 – Simulation de la transformation géométrique d'un rectangle au dessus du niveau du sol depuis deux points de vue différents de la caméra. À gauche la représentation 3D de la scène avec les deux positions du robot et la caméra. Les deux premiers graphiques à droite représentent les vues de la caméra dans la position au temps $(t-1)$ et (t). Le troisième graphique montre la projection de la première vue vers la deuxième (de $(t-1)$ vers (t)). Finalement le quatrième graphique montre la projection dans le sens inverse (de (t) vers $(t-1)$).

cupation. La première phase de l'algorithme pour créer cette grille est la projection des pixels vers le plan du sol. Il est nécessaire de refaire une transformation qui avait déjà été faite dans la transformation géométrique. Afin de réduire le temps calcul, nous avons reformulé la transformation géométrique en utilisant la transformation de perspective inversée (Inverse Perspective Mapping, IPM) qui a été développée par Mallot et al [101]. Cette transformation change l'angle de vue duquel une scène a été acquise en permettant de diminuer l'effet de perspective dans les images [128]. Le changement de l'angle de vue consiste à changer le domaine de perception des pixels vers un autre en 2 dimensions dont le contenu de l'information est distribué de façon plus homogène entre tous les pixels. Cela permet une mise en œuvre plus efficace des étapes de traitement suivantes [12]. L'application de cette transformation a besoin de conditions spécifiques d'acquisition (la position et l'orientation de la caméra, l'optique et autres) et d'une hypothèse sur la scène représentée dans l'image (dans notre cas défini comme une connaissance a priori) [119].

Cette approche divise notre première transformation géométrique en trois : les deux premières transformations sont les projections de chaque pixel des deux images au plan du sol et la troisième consiste en

une transformation due à la translation et à la rotation. L'équation 3.83 présente ces transformations qui ne dépendent pas de la position du robot ou du temps d'acquisition : ces deux transformations sont donc les mêmes. Dans l'équation 3.83, $\widehat{\mathbf{p}}^s$ représente les coordonnées homogènes du pixel $\widehat{\mathbf{p}}^i$ sur le plan du sol, c'est-a-dire, le sous-index *s* indique les coordonnées sur le sol tandis que *i* indique les coordonnées sur l'image. Finalement, il faut considérer la translation et la rotation de l'image acquise au temps $(t-1)$ déjà projetée sur le sol. Ces deux mouvements sont mesurés en fonction du déplacement du robot entre les acquisitions de l'image $(t-1)$ vers l'image (t) par la centrale inertielle du robot. L'équation 3.84 calcule les coordonnées estimées du pixel dans l'image au temps (t) en ajoutant la transformation odométrique du robot aux coordonnées $\widehat{\mathbf{p}}^s\,(t-1)$ au temps précédent.

$$\widehat{\mathbf{p}}^s = \mathbf{H}^{RC}\mathbf{H}^{CF}\mathbf{K}^{-1}\widehat{\mathbf{p}}^i \tag{3.83}$$

$$\widetilde{\mathbf{p}}^s\,(t) = \mathbf{H}^{ODO}\,(t-1,t)\,\widehat{\mathbf{p}}^s\,(t-1) \tag{3.84}$$

La détection d'obstacles pour cette méthode est faite en comparant le pixel courant avec le pixel estimé : si la différence dépasse un certain seuil, alors ce pixel a un niveau différent du sol et il est donc un obstacle. Un inconvénient dans cette méthode au moment de fusionner les deux méthodes de détection d'obstacles est le fait que s'ils sont exprimés dans des domaines différents, alors il faut changer de domaine dans la première méthode de détection d'obstacles. Pour résoudre ce problème nous appliquons l'équation 3.83 sur l'image de détection d'obstacles obtenue depuis la détection d'obstacles basée sur l'apparence.

3.4 Grille d'occupation

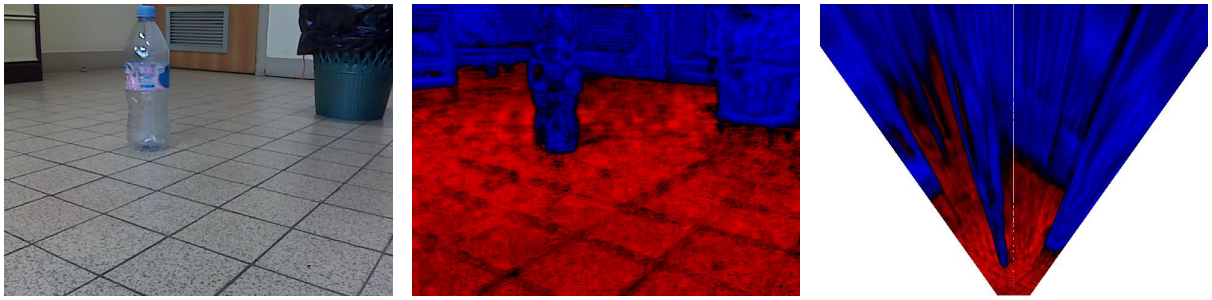
La dernière étape dans notre approche de détection d'obstacles est la création d'une grille d'occupation. Cette grille est une représentation multidimensionnelle stochastique qui contient les estimations des positions des obstacles dans l'environnement de navigation du robot [39]. De façon plus spécifique, une grille d'occupation est une représentation probabiliste mosaïque de l'information spatiale de l'environnement perçu par le robot pendant sa navigation [40].

Une grille d'occupation est composée de cellules qui représentent une partie de l'environnement de navigation du robot. La variable d'état $s\,(C)$ associée à une cellule C de la grille d'occupation est définie comme une variable aléatoire discrète à deux états, occupée et libre, notée O et L. La composition progressive de la grille d'occupation se fait de manière séquentielle en utilisant le théorème de Bayes pour déterminer la probabilité d'occupation d'une cellule. Soit $P\left[s\,(C_i) = O|\,\{\widetilde{\mathbf{p}}^s\}\,(t-1)\right]$ l'estimation actuelle d'état donnée d'une cellule C_i basée sur des observations $\{\widetilde{\mathbf{p}}^s\}\,(t-1) = \{\widetilde{\mathbf{p}}^s\,(1)\,,\widetilde{\mathbf{p}}^s\,(2)\,,...,\widetilde{\mathbf{p}}^s\,(t-1)\}$ et une nouvelle observation $\widetilde{\mathbf{p}}^s\,(t)$. La nouvelle estimation est donnée par l'équation suivante :

$$P\left[s\,(C_i) = O|\,\{\widetilde{\mathbf{p}}^s\}\,(t)\right] = \frac{p\left[\widetilde{\mathbf{p}}^s\,(t)\,|s\,(C_i) = O\right]P\left[s\,(C_i) = O|\,\{\widetilde{\mathbf{p}}^s\}\,(t-1)\right]}{\sum_{s(C_i)} p\left[\widetilde{\mathbf{p}}^s\,(t)\,|s\,(C_i)\right]P\left[s\,(C_i)\,|\,\{\widetilde{\mathbf{p}}^s\}\,(t-1)\right]} \tag{3.85}$$

où l'estimation préliminaire de l'état de la cellule $P\left[s\,(C_i) = O|\,\{\widetilde{\mathbf{p}}^s\}\,(t-1)\right]$. Cette estimation sert pour la connaissance a priori et est obtenue depuis la grille d'occupation. Initialement, l'estimation de la probabilité d'occupation d'une cellule est inconnue, c'est à dire sans connaissance a priori. L'estimation d'occupation pour une cellule $p\left[\widetilde{\mathbf{p}}^s\,(t)\,|s\,(C_i) = O\right]$ est donnée par la probabilité que le pixel projeté au sol qui coïncide avec la cellule C_i soit un obstacle. Cette estimation est obtenue en projetant l'image de la détection d'obstacles au sol. La figure 3.28 illustre le processus de calcul de cette estimation. D'abord, il faut calculer l'image de détection des obstacles par attributs de la couleur et de la texture (voir figure 3.28b), puis cette image est projetée au sol en utilisant les équations 3.83 te 3.84 (voir figure 3.28c).

Cette image, projetée sur le sol mais dans le repère du robot, est appelée grille d'occupation robot-centrée. Toutefois, si nous voulons construire une représentation plus générale de l'environnement, il est possible de changer la matrice de transformation $\mathbf{H}^{ODO}(t-1,t)$ de l'équation 3.84 par une matrice de transformation par rapport au monde $\mathbf{H}^{MR}(t-1)$, dans ce cas on parle de grille d'occupation centrée.



(a) (b) (c)

FIGURE 3.28 – Projection vers le sol de la détection d'obstacles obtenue à partir des attributs de la couleur et de la texture. Cette image a été acquise depuis une scène comprenant une bouteille en plastique et une poubelle qui sont considérées comme des obstacles pour la méthode de détection, (a) l'image originale, (b) l'image de probabilité de détection d'obstacles et (c) sa projection correspondante vers le sol.

La figure 3.29 montre la grille d'occupation centrée obtenue à partir des images de probabilité des obstacles qui sont calculées par les attributs de la couleur et de la texture. Dans la figure nous pouvons voir trois des trente trois scènes utilisées pour faire la construction de la grille d'occupation. Ci-dessous les images de la détection des obstacles projetées au sol. En dessous de ces images, la grille d'occupation centrée : les cellules noires sont les cellules occupées et les cellules blanches sont les cellules libres.

3.5 Conclusions

Nous avons présenté une approche pour détecter les obstacles d'une scène à partir de deux méthodes indépendantes, mais qui peuvent être traitées en parallèle. La première méthode calcule les attributs de la couleur et de la texture à partir d'une seule image couleur. Dans cette méthode les attributs de la couleur sont calculés en utilisant l'espace de couleur CIE-Lab, car cet espace présente une grande immunité au changement d'illumination. Les composantes chromatiques a et b de cet espace sont utilisées comme des attributs de couleur durant la classification, tandis que la luminosité L^* est utilisée pour calculer les attributs de texture, car elle offre le plus grand contraste. Les attributs de texture sont obtenus par l'adéquation des histogrammes de sommes et différences. L'avantage d'utiliser l'adéquation des histogrammes de sommes et différences est que la mémoire nécessaire pour stocker les histogrammes n'est plus nécessaire, ce qui est essentiel dans la mise en œuvre des algorithmes sur des systèmes embarqués.

Une base d'images est nécessaire pour créer une base de données avec les attributs de la couleur et de la texture. Cette base de données doit être étiquetée pour faire l'entraînement en mode supervisé. Plusieurs techniques d'apprentissage supervisé ont été évaluées pour trouver la méthode la plus appropriée dans notre approche de détection d'obstacles. Ces évaluations ont montré que l'algorithme AdaBoost discret a la meilleure performance globale et la meilleure performance pour détecter le sol. Ainsi, c'est cet algorithme que nous avons retenu pour la détection des obstacles. Les autres évaluations ont été élaborées dans le but de régler les différents paramètres dans le calcul des attributs pour la classification des objets dans une scène. Un inconvénient de cette technique est la phase d'apprentissage hors ligne qui peut être complétée par un apprentissage en ligne. Toutefois, cette phase d'apprentissage nécessite plus

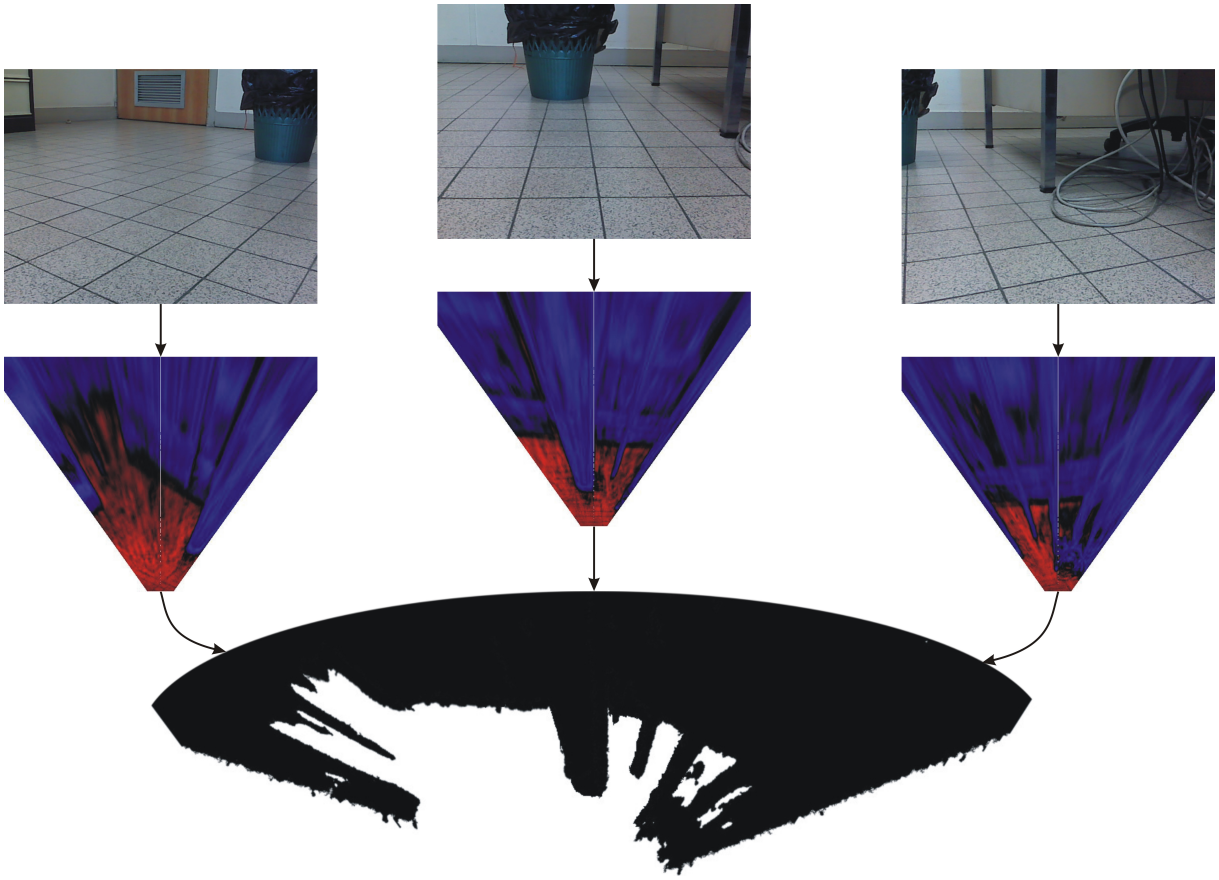


FIGURE 3.29 – grille d'occupation centrée en deux dimensions pour la navigation de un robot.

de puissance de calcul, mais elle permet de renforcer les connaissances acquises et d'améliorer ainsi la classification.

Un autre inconvénient de cette technique est le fait qu'une feuille de papier par terre est détectée comme un obstacle alors qu'elle ne représente pas de danger pour la navigation du robot. Pour cette raison nous avons développé une deuxième méthode de détection des obstacles.

La deuxième méthode de détection d'obstacles résout le problème de la feuille de papier par terre. Cette technique utilise deux images successives et l'information d'odométrie entre ces images pour calculer la hauteur des objets dans la scène. Ainsi la méthode est capable de détecter les objets qui peuvent présenter un risque pour la navigation du robot et donc les classer comme obstacles. Pour clore le chapitre, la méthode de la construction de la grille d'occupation est décrite.

Deuxième partie

Implémentation matérielle

Chapitre 4

Méthodologie et outils pour le développement matérielle

Sommaire

4.1 Introduction

Dans ce chapitre nous présentons d'une part les principes et architectures des systèmes à base de logique programmable et d'autre part les outils de conception qui leurs sont associés. Dans un premier temps nous expliquerons brièvement le principe de la logique programmable, et nous mettrons plus particulièrement en perspective le positionnement présent et futur de cette technologie par rapport à ses concurrentes : les circuits spécialisés de type ASIC, les processeurs à mot d'instructions très long (VLIW), les processeurs de signaux numériques (DSP) et les processeurs programmables.

Nous nous intéresserons ensuite à l'intégration «système» mettant en oeuvre ce type de ressources et présenterons les différentes techniques permettant d'associer logique reconfigurable et processeur programmable. Nous présenterons enfin le flot de conception utilisé pour ce type de technologie en insistant

plus particulièrement sur les outils de synthèse de haut niveau. Pour conclure le chapitre nous ferons une comparaison entre le flot de conception «classique» et le flot de conception utilisant des outils de synthèse de haut niveau.

4.2 Principes et architecture des circuits programmables

Le principe de la logique programmable remonte au début des années 1960, le concept ayant été proposé par G. Estrin [42, 41]. Mais ce n'est qu'au debut des années 1980 que les premières réalisations matérielles sont introduites sur le marché. L'apparition de ce type de circuit s'est d'abord faite au travers de circuits logiques programmables simples tels que les PAL (Programmable Array Logic) qui se programment comme des mémoires non volatiles de type ROM et sont utilisés pour implémenter des fonctions combinatoires simples (décodeurs d'adresse, contrôleurs de bus,...).

Avec les évolutions successives de la micro-électronique, différentes familles de circuits programmables ont vu le jour : les CPLD (Complex Logic Programmable Device) puis les FPGA (Field Programmable Gate Arrays), introduits par la société Xilinx en 1985. L'évolution des FPGA peut se ramener à quatre périodes.

La première allant des années 1985 à 1991 approximativement correspond à leur invention. A cette époque les FPGA étaient très petits, le nombre des applications très limité, la conception automatique était secondaire et la programmation se faisait à partir d'équations logiques ou de schémas graphiques simples [136].

La deuxième période, allant des années 1992 à 1999, correspond à une phase d'expansion avec l'apparition de circuits de plus en plus performants et reprogrammables à volonté, l'industrialisation et la commercialisation de ce type de circuit se faisant à grande échelle. La taille des FPGA se rapproche de la taille des systèmes à concevoir. Les premières applications en communications et traitement du signal et images commencent à apparaître. Des outils de conception HDL (Hardware Description Language) sont introduits et simplifient considérablement les phases de conception et simulation [113].

Durant la période 2000 à 2007 les circuits ASIC sont délaissés par les utilisateurs occasionnels au profit des FPGA dont le cycle de conception/fabrication est beaucoup plus simple et moins coûteux. On assiste alors à une expansion de cette technologie. La taille des circuits dépasse largement les besoins usuels et le coût, les normes, la consommation d'énergie devenant des facteurs déterminant dans la conception, la mise en œuvre et l'intégration de systèmes complets devient possible. Les performances de la logique reprogrammable sont limitées par le nombre d'entrées/sorties, la bande passante des mémoires et des cellules logiques. Parallèlement la conception de systèmes numériques complets exige de nouvelles compétences pour leur mise en œuvre [136, 113].

La période actuelle correspond à une phase de spécialisation au cours de laquelle la taille des FPGA dépasse la taille requise par la plupart des besoins. Cette évolution voit s'ouvrir de nombreux champs d'applications tels le traitement des signaux et des images, la sécurité et bien d'autres. La logique programmable peut être configurée une fois pour toutes ou de manière dynamique et le nombre d'outils de synthèse de haut niveau s'accroît considérablement [136, 113].

4.2.1 Ressources de traitement

Nous nous proposons d'aborder plus en détail, au cours du prochain paragraphe, l'architecture interne des circuits logiques programmables de type FPGA. Nous insisterons plus particulièrement sur les différents compromis qui lient le coût en surface du circuit, ses performances et la complexité des outils logiciels qui lui sont associés.

L'architecture d'un FPGA se décompose en deux types de ressources :

– Les ressources de traitement incluant les mémoires, la logique, et les registres. Elles sont regroupées en blocs logiques de différents types.
– Les ressources d'interconnexions programmables qui relient les blocs logiques entre eux.

La programmation d'un circuit reconfigurable consiste donc à spécifier la fonctionnalité de chaque bloc logique et à organiser le réseau d'interconnexion afin de réaliser la fonction demandée.

Dans le cadre de ce travail, nous nous intéresserons aux architectures de circuits programmables de type matricielles. Dans ce type de circuits, les blocs logiques sont organisés en une structure rectangulaire régulière, chaque bloc étant relié au réseau de routage grâce à des Points d'Interconnexion Programmables (PIP). Le réseau de routage quant à lui, est formé de canaux horizontaux et verticaux qui occupent l'espace séparant les blocs logiques comme illustré par la figure 4.1. La programmabilité du routage est assurée par des matrices d'inter-connexion, placées à chaque croisement de canaux.

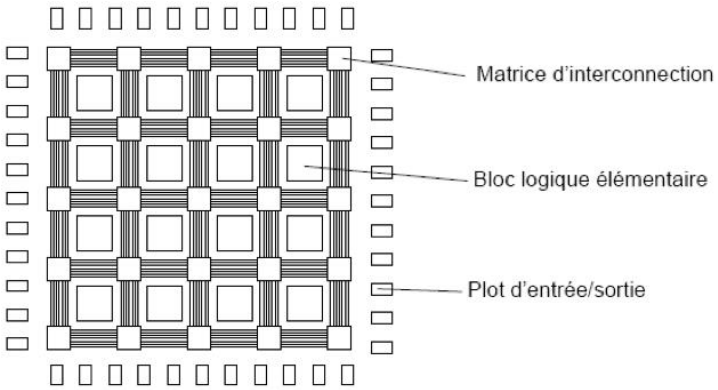


FIGURE 4.1 – Vue simplifiée d'un circuit programmable de type FPGA à organisation matricielle et ses différents composants internes.

Blocs logiques

Il est très important de bien choisir l'architecture du bloc logique. En effet ce choix déterminera le type de fonctionnalité supporté par le circuit reconfigurable. Les blocs logiques se distinguent en général d'après leur granularité (en d'autre termes le niveau de complexité intégrable dans un seul bloc). Généralement ils sont répertoriés en trois catégories :

– Les blocs logiques à grain fin : ceux-ci sont composés d'un assemblage de portes logiques de type ET, OU, NON, permettant l'émulation de fonctions logiques simples (à 2 ou 3 entrées), ou des éléments de mémorisation (registres ou bascules) [22] ; (un exemple est présenté sur la figure 4.2a).
– Les blocs logiques à grain moyen : ils sont conçus à partir de fonctions sous forme de tables programmables (Look Up Tables), combinées à un ou plusieurs éléments de mémorisation. La taille des tables utilisées autorise généralement entre 8 et 32 adresses d'entrées différentes (suivant le circuit utilisé) avec un résultat sur un ou plusieurs bits. Chacune de ces tables permet ainsi d'émuler une fonction logique quelconque à 3, 4 ou 5 entrées [23, 32, 31], (un exemple est présenté sur la figure 4.2a).
– Les blocs logiques à gros grain : généralement ils sont construits autour d'un opérateur arithmétique et/ou logique (pouvant effectuer des opérations d'addition, de multiplication et des opérations logiques au niveau bit), combiné à un ensemble de registres organisés en file. Ce type d'architecture est très proche d'un chemin de données de processeur programmable [100, 103] (un exemple est présenté sur figure 4.2b).

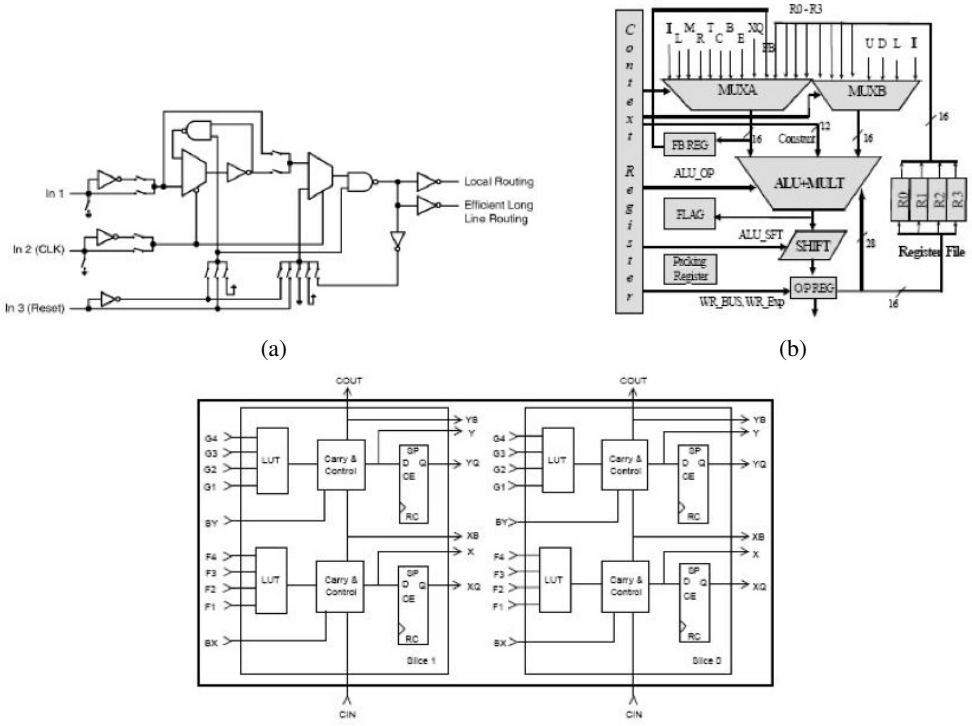


FIGURE 4.2 – Architectures de différents blocs logiques (a) bloc logique à grain fin utilisé dans le circuit ProAsic de la société Actel composé de simples portes logiques ; (c) bloc logique à grain moyen du FPGA Virtex de la société Xilinx composé de quatre look-up-tables ainsi que de ressources dédiées au traitement arithmétique (génération de retenue) ; (b) bloc logique du circuit Morphosys composé d'une ALU et de registres.

Le choix de la granularité implique un compromis entre flexibilité et performance. Une architecture à grain fin permet d'implanter une plus grande variété de circuits puisque les blocs logiques peuvent émuler des fonctions logiques quelconques. Cependant cette versatilité se fait aux dépens des performances : un opérateur arithmétique implanté sous la forme d'une combinaison de blocs logiques à grain fin sera moins rapide, utilisera plus de ressources et dissipera plus d'énergie qu'un bloc à gros grain. En revanche, l'implantation d'une machine à états complexe ou d'opérateurs arithmétiques inhabituels (arithmétique redondante, corps de Galois, etc.) ne sera pas toujours efficace sur ces derniers. C'est pour toutes les raisons citées ci-dessus que les circuits programmables modernes offrent désormais des architectures hétérogènes pour les ressources de traitement comme pour les ressources de mémorisation.

Les architectures de blocs logiques de type grain moyen intègrent désormais des ressources spécialisées permettant une gestion efficace des opérations arithmétiques et logiques [31]. Ces ressources comprennent en général une part de logique pour la génération/propagation de retenues, combinée avec des ressources de routage dédiées. Par ailleurs, certaines familles de FPGA intègrent désormais des blocs logiques de type multiplieurs [24, 33, 34, 25, 30], lesquels évitent l'implémentation de ces d'opérateurs à partir de blocs logiques de type LUT, ces derniers engendrant généralement des structures lentes et coûteuses en surface.

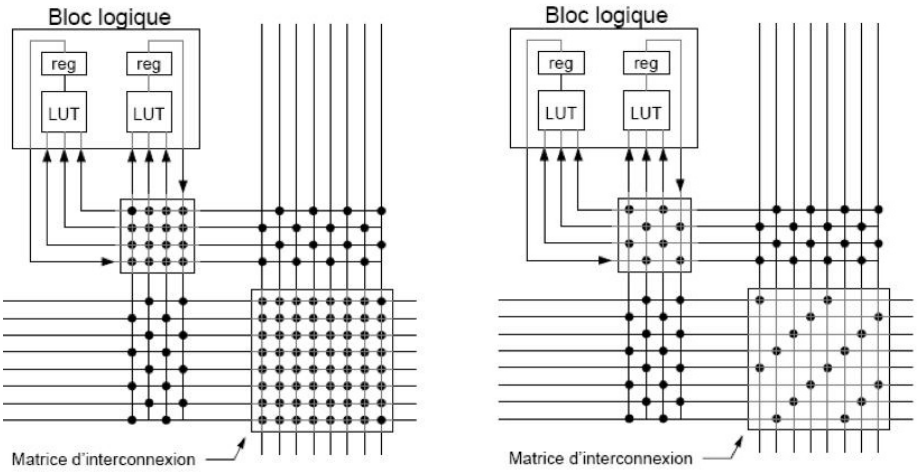
Les ressources de mémorisation bénéficient également de cette tendance à l'hétérogénéité. La plupart des familles de circuits récentes intègrent désormais des ressources mémoires à grain moyen (connues sous le terme de Blocs Mémoire Embarqués) qui offrent des capacités de quelque kilo-bits et une inter-

face (nombre de lignes d'adresses et largeur du chemin de données) configurable [32, 23, 25, 30]. Par ailleurs, certaines architectures permettent d'utiliser les LUT présentes dans les blocs logiques comme des mémoires à grain fin (SRAM asynchrone, ou bien encore ligne à retard synchrone programmable). De fait, la combinaison de ces différents types de ressources permet la construction d'un système hiérarchisé de mémoire distribuée dont l'organisation est spécifiée par le concepteur en fonction de l'application à implanter.

4.2.2 Le routage

Routage programmable classique

Comme nous l'avons vu précédemment, les ressources de routage consistent en un ensemble de canaux séparant les blocs logiques entre eux. Ces canaux bénéficient de deux types de connexions : soit directement connectés aux blocs logiques, soit reliés entre eux au travers de matrices d'interconnexions. Le rôle de la matrice d'interconnexion est d'aiguiller les signaux provenant des différents canaux. Cet aiguillage se fait à l'aide d'un système d'interconnexion programmable. Le choix de l'organisation de ce système induit un compromis entre la souplesse du réseau d'interconnexion et le coût en surface. Si le choix offrant le plus de flexibilité est celui d'un cross-bar complet (voir figure 4.3a), il engendre un coût en surface important (le nombre de transistors nécessaires à l'implémentation du cross-bar évolue de manière quadratique avec la largeur des canaux). Afin de réduire ce coût, on utilise en général un réseau cross-bar dépeuplé (voir figure 4.3b) qui offre une flexibilité réduite au bénéfice d'un nombre de transistors de commutation plus faible.



(a) (b)

FIGURE 4.3 – Différentes structures de matrices d'interconnexions (a) réseau cross-bar complet (b) réseau cross-bar dépeuplé d'un facteur 4.

Dans ces approches, l'organisation des ressources de routage est régulière et dans le meilleur des cas, le nombre de matrices d'interconnexion à traverser pour connecter deux blocs logiques entre eux est proportionnel à la distance de Manhattan qui les sépare. Le délai d'interconnexion, proportionnel au nombre de transistors traversés par le chemin d'interconnexion, évolue donc également linéairement avec cette distance. Ce type de routage engendre des délais d'interconnexion très pénalisant lorsqu'il s'agit de relier des blocs logiques éloignés. Avec l'apparition de FPGA de plus en plus denses, ce type de routage se révèle donc de moins en moins efficace à la fois en termes de surface et de performance.

Routage segmenté

Afin de remédier à ce problème, des architectures de routages dites segmentées ont été proposées [13]. Pour ce type de réseau de routage (figure 4.4a), les canaux d'interconnexion sont de longueur variable et ne passent donc pas par toutes les matrices d'interconnexion présentes sur leur chemin. Une telle approche présente deux avantages :

– Le nombre de transistors traversés n'évolue plus linéairement avec la distance de Manhattan entre blocs logiques, et de ce fait, les délais d'interconnexion entre blocs logiques éloignés ne dégradent plus autant les performances du circuit.
– Le fait que certaines lignes de routage évitent les matrices d'interconnexion présentes sur leur chemin, permet de réduire de façon sensible la taille de ces matrices ; celles-ci utilisent donc un nombre réduit de transistors, permettant ainsi une meilleure densité d'intégration.

Si cette approche permet des gains très attractifs en termes de performances matérielles, elle rend cependant le problème du routage beaucoup plus complexe en obligeant les outils à gérer des ressources de routages hétérogènes.

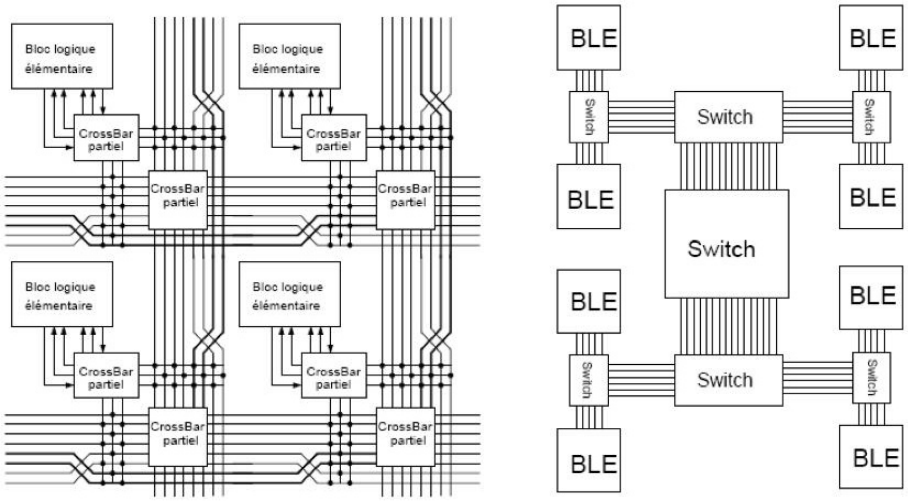


FIGURE 4.4 – Deux types de routages évolués : (a) un exemple de réseau de routage segmenté avec des longueurs de canaux de 1 et 2 ; (b) un exemple de routage hiérarchique dans lequel le réseau est organisé en arbre..

Routage hiérarchique

Une autre approche propose une organisation hiérarchique du réseau d'interconnexion (figure 4.4b) : celui-ci est organisé de façon à ce que les délais d'interconnexion évoluent linéairement avec le nombre de couches traversées par une connexion [137].

L'idée s'inspire du caractère hiérarchique des circuits logiques qui sont généralement organisés en un arbre de modules et de sous-modules. En regroupant les blocs logiques appartenant à un module donné au sein du même niveau hiérarchique, il doit être possible d'exploiter la localité des interconnexions dans le module et ainsi réduire les besoins en ressource de routage au niveau des matrices d'interconnexions.

Il faut cependant parvenir à réaliser une partition du circuit adaptée au réseau d'interconnexion. Cette tâche peut s'avérer difficile dans la mesure où les différents blocs du circuit ne partagent pas forcément les mêmes caractéristiques en termes d'utilisation de ressources et de besoin d'interconnexions.

4.2.3 Positionnement technologique

Les circuits programmables se présentent comme une alternative à une solution entièrement logicielle ou une solution matérielle dédiée, basée sur l'utilisation d'un circuit VLSI. Cependant il est important de positionner cette technologie par rapport à ses concurrentes, au moins sur les critères les plus courants, à savoir : la performance, le temps de développement, le coût, et le niveau de consommation électrique.

Performances

D'un point de vue empirique, on s'aperçoit qu'un circuit réalisé en logique programmable possède en général une fréquence de fonctionnement entre trois et cinq fois moindre qu'une réalisation sur un circuit spécialisé pour une technologie de gravure équivalente.

Cette différence de performance vient directement de la nature programmable de ces circuits. Elle varie cependant fortement avec l'application choisie, et plus encore avec les choix architecturaux. L'utilisation d'architectures optimisées pour la logique programmable permet souvent de réduire cet écart, et des réalisations particulières parviennent même à obtenir de meilleures performances que des solutions ASICs équivalentes [117]. De telles réalisations restent toutefois des exceptions. Enfin, il faut également noter que, grâce aux efforts des constructeurs, les circuits programmables sont souvent disponibles pour des technologies de gravures plus fines que la plupart des ASIC. À titre d'exemple les circuits reconfigurables les plus récents sont produits avec des technologies de gravures de $0.15\ \mu m$. Ce n'est pas le cas pour les ASIC, dont la majorité de la production se base encore, pour des raisons de coût de conception et de production, sur des technologies en $0.25\ \mu m$.

La comparaison avec des processeurs programmables est beaucoup plus délicate. Par nature, une implémentation logicielle est très différente d'une implémentation matérielle. Si les fréquences de fonctionnement des processeurs programmables sont en général très supérieures à celles d'un circuit programmable (par exemple $1,7$ GHz pour un Pentium III contre 200 MHz pour les familles de circuits les plus rapides), une implémentation sur logique programmable permet en général de tirer parti d'un fort degré de parallélisme lequel compense en général très largement leur fréquence d'horloge plus faible. Ainsi en fonction de l'application et de la densité du circuit, on peut espérer obtenir des gains en performance d'un, voire deux ordres de grandeur [95, 94].

Consommation

En ce qui concerne le critère de consommation électrique, encore une fois, la solution logique programmable se présente comme une alternative intermédiaire entre ASIC et processeur programmable. L'énergie dissipée par une solution réalisée en logique programmable est en général supérieure d'au moins un ordre de grandeur à celle d'une solution VLSI classique (cette règle empirique dépend toutefois très fortement de la granularité d'architecture du circuit comme expliqué dans la section 4.2). Ici encore, c'est la nature programmable du circuit, et en particulier le surcoût associé en termes de ressource silicium, qui engendre cet écart.

De même, on peut constater des différences de consommation d'un ordre de grandeur entre une solution logicielle sur DSP et son équivalent sur FPGA. Ceci est encore dû à la possibilité pour les FPGA de bénéficier d'une architecture spécialisée, et de fonctionner à une fréquence d'horloge beaucoup plus faible en bénéficiant d'un fort degré de parallélisme [6].

Coût

Pour une densité utilisable donnée, leur programmabilité rend ces circuits plus gourmands en ressource silicium qu'un circuit VLSI classique. De fait, le coût en production à l'unité d'un ASIC est très

attrayant par rapport à celui d'un circuit programmable. Cette différence de coût doit cependant être relativisée car le coût total d'un circuit comprend également une partie incompressible (conception, prototypage, production) indépendante du volume de production. Ces types de coûts, connus sous le terme de CNR (coût non récurrents) sont beaucoup plus faibles pour un circuit programmable, puisque, d'une part, leur conception est plus simple, et d'autre part, ils ne nécessitent pas la mise en place d'une chaîne de production spécifique.

C'est pourquoi les circuits programmables représentent une solution très économique lorsque les volumes de production sont faibles.

La comparaison avec les processeurs programmables est là encore, très délicate. En effet, les circuits programmables sont utilisés pour des systèmes nécessitant des performances dépassant souvent largement celles des processeurs programmables classiques. Comparer le coût d'un circuit reprogrammable avec celui d'un processeur de type DSP ou micro-contrôleur n'est donc pas vraiment pertinent. On peut cependant rapprocher ces performances de systèmes multi-processeurs qui peuvent en général rivaliser avec des circuits programmables de densité intermédiaire. Dans ce contexte le circuit reconfigurable se révèle en général beaucoup plus économique [52].

Temps de développement

Les solutions de type logique programmable offrent un avantage certain en ce qui concerne le temps de développement. Effectivement, la mise au point et la validation du fonctionnement d'un circuit programmable dans son environnement matériel se fait en général très rapidement : obtenir un circuit opérationnel à partir de sa spécification architecturale ne nécessite en général que quelques heures ; par contre pour un circuit VLSI dédié, cette étape requiert plusieurs semaines. Étant donné le nombre d'itérations entre la phase de conception et la phase de validation, le cycle de développement d'un circuit programmable s'avère donc largement plus court que pour un ASIC (quelques semaines contre plusieurs mois).

De plus, la conception du circuit en elle-même est simplifiée. Par exemple, l'intégration de systèmes de test et de déboguage in-situ évite d'avoir à concevoir le circuit en vue de sa testabilité. Ce gain est important : pour certains circuits, les ressources silicium dédiées à la testabilité peuvent représenter jusqu'à 30% de la surface du circuit, et représentent souvent un effort de conception supplémentaire non négligeable. La comparaison est en revanche beaucoup moins avantageuse lorsqu'il s'agit de processeurs programmables : il est bien connu que le principal avantage d'une solution logicielle est le gain de temps en conception. La mise au point d'une application qui peut prendre plusieurs semaines pour un FPGA peut en général s'effectuer en quelques heures sur un processeur programmable. Sur ce plan, les circuits FPGA sont, à l'heure actuelle, encore très loin de pouvoir rivaliser avec une solution logicielle. On peut cependant noter l'apparition d'outils de programmation à partir de langages ou environnements de haut niveau (C/C++, SystemC, Matlab ou LabVIEW) [35, 58, 21], qui permettent aujourd'hui de réduire les temps de conception, les rapprochant de ceux d'une solution logicielle. Ces gains se font toutefois au prix d'une réduction très sensible des performances du circuit obtenu.

4.3 Conception sur Matériel

Au cours de cette section, nous allons décrire brièvement le flot de conception classique d'une architecture basée sur des FPGA, puis le langage de description matériel et les outils de conception associés.

4.3.1 Flot de conception au niveau RTL

La figure 4.5 montre le flot de conception classique au niveau RTL (register transfer level). Ce flot de conception s'appuie sur les spécifications fonctionnelles et les contraintes ou les performances requises

par l'architecture à réaliser. La plupart des algorithmes ayant été développés pour être implémentés sur un ordinateur d'une manière séquentielle, il est donc nécessaire de faire l'adéquation de l'algorithme pour sa mise en œuvre sur un système reconfigurable [105, 113].

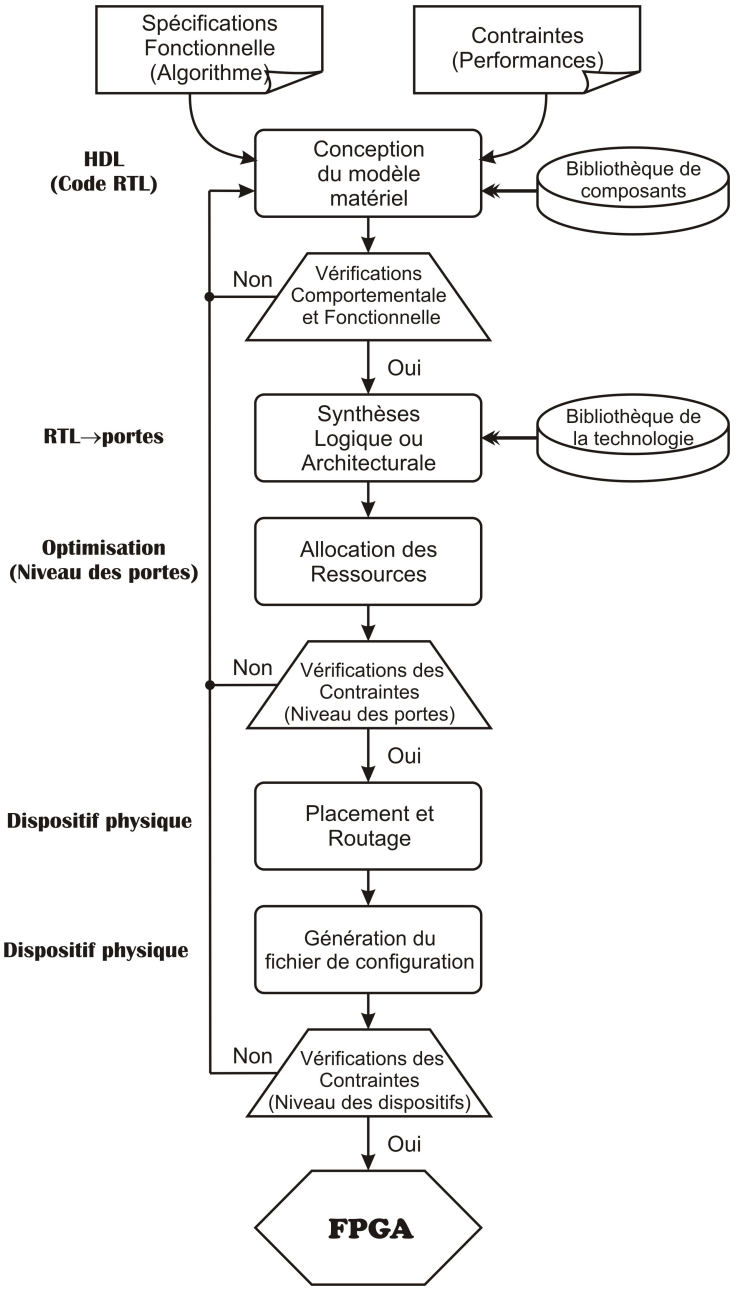


FIGURE 4.5 – Flot de conception au niveau RTL.

L'adéquation algorithmique est très importante dans la conception des architectures car elle nous permettra d'obtenir le meilleur compromis entre la performance et les ressources nécessaires et peut s'avérer cruciale lors de l'intégration sur FPGA. L'adéquation algorithmique la plus simple est une parallélisation des opérations recensées dans l'algorithme d'origine. Cette méthode est relativement simple, avec un temps de conception court, mais elle n'optimise pas les ressources consommées, et dans certains cas, les architectures développées peuvent ne pas respecter les performances requises. Une adéquation plus

complexe consiste en la remise en forme de l'algorithme en tenant compte des ressources disponibles dans le circuit et optimiser celles-ci pour obtener la performance requise. Cependant, cette dernière méthode prend beaucoup plus de temps et les architectures développées, propres au circuit, font que parfois l'algorithme perd de sa généralité et n'est plus exploitable dans d'autres applications voire sur un autre dispositif. Il est donc très important que l'adéquation conserve la généralité de l'algorithme tout en prenant en compte l'organisation interne des systèmes reconfigurables.

Afin d'obtenir une adéquation algorithmique plus efficace, nous divisons celle-ci en deux étapes : la première s'applique directement à l'algorithme, sans prendre en compte de dispositif particulier ; c'est à dire que l'algorithme est redéfini uniquement dans les buts de la parallélisation et de la réduction des ressources nécessaires à sa mise en œuvre. Cette approche permet la réutilisation de l'adéquation de l'algorithme dans d'autres dispositifs tels que les ASIC, GPU, DSP et en même temps, conserve la généralité de l'algorithme. Ainsi, les spécifications fonctionnelles pour le flot de conception sont plus faciles à obtenir et plus adaptées à la mise en œuvre sur les FPGA. Il faut souligner que cette première adéquation ne tient pas compte des contraintes requises pour le système à réaliser.

Dans la deuxième étape le flot de conception (voir figure 4.5) prend en entrée d'une part les spécifications fonctionnelles issues de la première étape et d'autre part les contraintes du système. Ce flot de conception sur architecture reconfigurable se décline en cinq étapes qui seront brièvement décrites dans les paragraphes suivants.

Conception du modèle matériel

Cette étape a pour but de fournir une description de l'architecture du circuit à implanter sous la forme d'un graphe de primitives logiques (portes ET, OU, NON, registres et bascules), à partir des spécifications fonctionnelles. Dans la conception des modèles matériels, l'adéquation algorithme-architecture joue un rôle très important. Cette adéquation contrairement à l'adéquation algorithmique, doit prendre en compte les avantages du traitement sur les systèmes reconfigurables. Cela dans le but de trouver une architecture fournissant le meilleur compromis entre performance et ressources consommées.

Les HDL (langages de description du matériel) comme VHDL [1] ou Verilog [135] sont les plus utilisés dans la conception des modèles matériels pour FPGA. Ces HDL utilisent des bibliothèques de composants qui contiennent des opérateurs de base tels que les portes ET, OU, NON, etc. jusqu'à des algorithmes complets tels que la FFT, les filtres FIR, etc., ces derniers étant disponibles sous forme d' IP (intellectuel property).

L'utilisation de ces IP peut considérablement réduire le temps de conception. Toutefois, il est bon de se rappeler que la plupart d'entre eux sont adaptés à un fabricant, une famille de circuits voire à un circuit particulier, ce qui limite la réutilisation de ces architectures. Inversement la conception à partir de cellules de base prend plus de temps et repose fortement sur l'expérience et l'expertise du concepteur. La méthode qui semble la mieux adaptée pour réduire le temps de conception est celle qui consiste à réutiliser des IP génériques, lesquelles facilitent la portabilité des architectures sur des circuits de diverses familles et divers fabricants. Il faut souligner cependant que la portabilité a un coût sur le temps de conception et la performance.

De manière générale l'architecture proposée est évaluée dans son comportement et fonctionnement puis comparée à d'autres architectures elligibles sur la base de critères prédéfinis (ressources consommées, consommation d'énergie,...).

Synthèse

La deuxième étape du flot de conception est la synthèse. Deux types de synthèse peuvent être distingués :

– La synthèse logique à partir d'une spécification structurelle qui décrit explicitement l'architecture comme un ensemble d'opérations combinatoires connectées au travers de registres,
– La synthèse comportementale dans laquelle l'utilisateur décrit le comportement de l'architecture à un plus haut niveau d'abstraction et où l'outil de synthèse génère une (ou plusieurs) architecture(s) satisfaisant les spécifications fournies par l'utilisateur.

Allocation des ressources

L'objectif de cette étape est d'associer à chaque primitive logique obtenue après synthèse, un bloc logique du circuit programmable. Il s'agit donc de regrouper les cellules logiques élémentaires de manière optimale car la qualité de l'allocation choisie a une influence à la fois sur les ressources nécessaires à l'implémentation du circuit et sur ses performances.

Dans la plupart des cas, l'allocation consiste à trouver, en fonction des contraintes fournies par l'utilisateur, le meilleur compromis entre les critères de vitesse et de surface en utilisant des techniques de réplication et/ou fusion des primitives logiques.

Placement et routage

Ces deux étapes prennent en charge le placement physique des blocs logiques puis la programmation du réseau d'interconnexion. Les problèmes du placement et du routage ne sont pas indépendants : la qualité du placement pouvant fortement influencer la routabilité du circuit.

Le compromis est ici entre localité et congestion : afin de permettre des interconnexions plus courtes, il est préférable de placer les blocs reliés entre eux, proches les uns des autres, ceci afin de minimiser le nombre de matrices d'interconnexion traversées. Toutefois, le placement d'un nombre élevé de blocs logiques fortement interconnectés, sur un espace limité, peut rapidement poser des problèmes de congestion. De fait, le réseau d'interconnexion n'est plus capable de router toutes les connexions efficacement et les délais s'en trouvent dégradés.

Génération du fichier de configuration

Une fois que la description complète de l'implémentation est obtenue, un fichier de configuration contenant l'ensemble des informations relatives à l'implémentation est généré. Celui-ci peut alors être utilisé pour la configuration du circuit programmable, soit à l'aide d'une mémoire non-volatile associée au circuit, soit directement à partir d'une interface externe (processeur, bus, liaison parallèle, etc.).

Une validation finale de l'architecture dans son contexte permettra de connaître la performance réelle du système. Si celle-ci est conforme aux contraintes requises, alors le flot de conception est terminé. Dans le cas contraire, il est nécessaire de revenir au début du flot de conception et d'ajuster l'architecture pour obtenir les performances souhaitées.

4.3.2 Langage de description matériel

Les HDL décrivent le comportement d'un circuit ou système électronique à partir duquel le circuit physique sera implémenté. Les HDL les plus utilisés sont le VHDL et le Verilog. Nous nous concentrerons sur le langage VHDL étant donné sa normalisation, son indépendance vis à vis de la technologie et des fabricants, sa portabilité et sa réutilisabilité [118, 36].

VHDL signifie VHSIC Hardware Description Language. VHSIC est lui-même une abréviation pour Very High Speed Integrated Circuits. VHDL est un langage de modélisation et de description conçu pour décrire la fonctionnalité et l'organisation des systèmes numériques, circuits et de composants.

VHDL, développé initialement comme un langage de modélisation et simulation logique contrôlé par des événements dans des systèmes numériques, est actuellement également utilisé pour la synthèse automatique de circuits. Le VHDL a été développé de manière très similaire au langage ADA, lequel a été choisi comme modèle car son organisation est orientée systèmes temps réel et matériel en général. [113].

VHDL a été développé sous l'impulsion du DoD (Departement of Defense) aux Etats Unis dans les années 1980. Ce langage de description est devenu le premier langage de description matériel à être normalisé par l'IEEE (Institute of Electrical and Electronics Engineers) sous la référence 1076 en 1987. Révisé en 1993 pour supprimer quelques ambiguïtés et améliorer la portabilité du langage, cette norme est vite devenue un standard en matière d'outils de description de fonctions logiques [3].

Les deux principales applications immédiates de VHDL sont les dispositifs logiques programmables (PLD, CPLD ou FPGA) et les circuits ASIC. Un fois le code VHDL écrit, celui-ci peut être utilisé soit pour mettre en œuvre le circuit dans un dispositif programmable ou pour fabriquer une puce ASIC. Actuellement, de nombreuses puces sont conçues en utilisant une telle approche [118].

La norme qui définit la syntaxe et les possibilités offertes par le langage de description VHDL est complète et flexible. Elle permet de faire de la modélisation structurelle et comportementale. VHDL permet une modélisation précise dans des styles différents, du comportement d'un système numérique connu et le développement de modèles de simulation.

La synthèse de circuits commence par une spécification des entrées avec un certain niveau d'abstraction et se termine par une mise en œuvre plus détaillée. Par conséquent, elle peut être vue comme une tâche verticale reliant les différents nivaux d'abstraction partant du haut et allant vers le bas de la hiérarchie. A l'origine le VHDL a été conçu pour la modélisation des systèmes numériques. C'est pour cette raison que son utilisation dans la synthèse n'est pas immédiate, mais l'apparition d'outils de synthèse sophistiqués permet la mise en œuvre d'architectures spécifiques décrites avec un haut niveau d'abstraction.

La synthèse à partir de VHDL est aujourd'hui largement répandue et il existe une forte demande pour son utilisation. Par ailleurs les outils de synthèse fondés sur ce langage offrent aujourd'hui des gains de productivité lors de la conception assez conséquents.

VHDL est destiné à la synthèse ainsi qu'à la simulation des circuits. Cependant, si VHDL est entièrement simulable, toutes les constructions ne sont pas synthétisables pour autant et il est important de s'en assurer lors du flot de conception au niveau RTL. Les avantages de l'utilisation de ce langage pour la description du matériel sont les suivantes :

- Conception de haut niveau : c'est à dire qui ne suit plus la démarche descendante habituelle (du cahier des charges jusqu'à la réalisation et le calcul des structures finales) mais qui se "limite" à une description comportementale directement issue des spécifications techniques du produit que l'on souhaite réaliser. Cependant, VHDL nous permet malgré tout de concevoir, modéliser et tester un système à partir d'un niveau d'abstraction élevé jusqu'au niveau de la définition structurelle des portes.
- Synthèse sur du matériel : Les circuits décrits en VHDL, moyennant quelques directives pour la synthèse, peuvent être implémentés à partir d'outils de synthèse produisant une description niveau portes.
- Langage normalisé : étant basé sur un standard, les ingénieurs concepteurs peuvent utiliser ce langage pour minimiser les erreurs de communication et les problèmes de compatibilité.
- Portabilité des descriptions VHDL : possibilité de cibler une description VHDL dans le composant ou la structure que l'on souhaite en utilisant l'outil que l'on veut.
- Conception Top-Down : c'est à dire, pour décrire (modéliser) le comportement des blocs de haut niveau, analyser (simuler) et affiner leurs fonctionnalités avant d'atteindre des niveaux plus bas d'abstraction de la mise en œuvre de la conception.

– Modularité des composants : permet de diviser ou de décomposer une conception matérielle et sa description VHDL en unités plus petites.

La description comportementale et structurelle

Il y a deux façons de décrire un circuit. D'un côté, on peut le décrire en indiquant les différentes composantes qui le forment ainsi que leurs interconnexions afin de répondre à un fonctionnement souhaité. C'est la forme habituelle utilisée et elle met en oeuvre des outils de saisie de schémas et de création de netlist. La deuxième façon est de décrire un circuit en indiquant l'évolution de son comportement en regard d'évènements. Bien sûr, cette façon de faire est beaucoup plus agréable pour un concepteur car ce qui lui importe vraiment c'est le fonctionnement du circuit plutôt que sa constitution. A contrario il sera plus difficile de mettre en oeuvre ce circuit décrit par son seul comportement.

Le VHDL est intéressant car il permet les deux types de description :

– **Structurel :** il peut être utilisé comme un langage ordinaire de Netlist qui précise d'une part les composants du système et d'autre part leurs interconnexions.
– **Comportemental :** il peut également être utilisé pour la description comportementale ou fonctionnelle d'un circuit. C'est la différence avec un langage de type netlist. Sans connaître la structure interne d'un circuit il est possible d'en décrire son fonctionnement. Ceci est particulièrement utile dans la simulation car il devient possible de simuler un système sans connaître sa structure interne.

4.3.3 Outils de conception

Dans cette section nous décrivons brièvement les outils utilisés pendant la conception matérielle. Si nous revenons au flot de conception au niveau RTL (voir figure 4.5), la première étape concerne la conception du modèle matériel, lequel doit être validé au niveau fonctionnel. L'outil de simulation ModelSim a été utilisé pour cette étape. Modelsim est un simulateur HDL développé par Mentor Graphics qui supporte plusieurs langages (VHDL, Verilog, SystemVerilog et SystemC) [75]. Nous avons choisi ce simulateur car il est largement utilisé dans le groupe de recherche et il permet de ré-utiliser de nombreux modules développés dans des langages hétérogènes. ModelSim fournit également un environnement complet de simulation et de débogage pour les designs complexes intégrés dans des FPGA ou ASIC.

Bien que ModelSim puisse être utilisé dans presque toute la chaîne de conception, la synthèse sur un composant cible particulier est nécessaire pour la mise en œuvre. Par conséquent, nous avons utilisé l'environnement de conception Quartus II développé par Altera [28]. Il convient de souligner que ces modules sont conçus pour être les plus portables possible (sans technologie), jusqu'à la phase d'incorporation dans le système. Ils peuvent donc être mis en oeuvre sur des circuits FPGA de divers fabricants.

Dans le cadre de l'environnement de développement Quartus II, le SOPC Builder est un outil de développement de systèmes sur puce. Il permet de définir et générer un système complet sur une puce programmable (SOPC) en moins de temps que par des méthodes traditionnelles d'intégration manuelle [29].

SOPC Buider automatise la tâche d'intégration des composants matériels. Dans les méthodes de conception traditionnelles les modules sont décrits manuellement au «fil de l'eau» puis rassemblés pour constituer le système. Comparativement dans SOPC Builder, on spécifie les composants du système via une interface graphique (GUI) et celui-ci génère l'interconnexion logique automatiquement. Il gère les fichiers HDL qui définissent tous les composants du système et génère un fichier HDL de type VHDL ou Verilog de haut niveau qui relie tous les éléments ensemble [29].

Un module généré par SOPC Builder peut être automatiquement intégré dans un autre système lui-même généré par SOPC Builder. On peut définir et ajouter des composants personnalisés ou choisir

parmi une liste de composants fournis. L'utilisation de ces librairies de composants simplifie la conception puisque beaucoup d'entre eux sont d'usage courant. On trouve notamment des interfaces de bus, des contrôleurs, des processeurs ou des fonctions de traitement général lesquels réduisent le temps de conception.

Pour qu'un module puisse être intégré au SOPC Builder, il doit respecter le protocole Avalon (memory-mapped, streaming et/ou IRQ) pour la connexion physique des composants. On peut utiliser le SOPC Builder pour connecter n'importe quel périphérique logique (sur la puce ou hors puce) qui a une interface Avalon. Il existe différents types d'interfaces Avalon comme il est décrit dans les spécifications associées (voir la référence [27] pour plus de détails).

La figure 4.6 illustre un système sur FPGA comprenant une partie générée par SOPC Builder et des modules logiques personnalisés. Il est possible d'intégrer une logique personnalisée à l'intérieur ou à l'extérieur du système SOPC Builder. Dans cet exemple, le composant personnalisé à l'intérieur du système SOPC Builder communique avec d'autres modules à travers une interface Avalon-MM Master. La logique personnalisée à l'extérieur du système SOPC Builder est connectée au système via une interface PIO (Programmed Input-Output). Le système SOPC Builder comprend deux composants issus de la librairie et connectés via une interface Avalon-ST source et sink.

Un composant peut être un dispositif logique entièrement contenu dans le système SOPC Builder, tel celui montré dans la figure 4.6. Alternativement, un composant peut agir comme une interface à un dispositif externe (hors puce) tel que le composant d'interfaçage avec la mémoire DDR2. En plus de l'interface Avalon, un composant peut avoir d'autres signaux connectés à la logique externe au système SOPC Builder. Les signaux non-Avalon peuvent fournir une interface spécifique au système SOPC Buider, tels que la PIO.

La figure 4.7 illustre un composant SOPC et les interfaces de communication au protocole Avalon. L'interface Avalon-ST est utilisée pour le traitement de flux de données, tels que le flux vidéo. Dans notre cas, les modules de traitement des images ont en entrée le protocole Avalon-ST Sink et en sortie le protocole Avalon-ST Source. Si les modules ont des registres de configuration ou s'ils ont besoin de lire dans des mémoires, ils utilisent le protocole Avalon-MM Slave. Dans le cas où ils ont besoin de stocker des données dans les mémoires, ils utilisent une interface Avalon-ST Master. Il est possible également que les modules aient une interface externe pour communiquer avec des périphériques externes (caméras, vga, ethernet, etc.).

Pour préserver la portabilité des modules développés, nous avons séparé le protocole Avalon des modules de traitement afin que ceux-ci aient un protocole générique. Chaque module de traitement est contenu dans un module qui convertit le protocole de communication générique au protocole Avalon. Ainsi, les modules peuvent être intégrés directement dans le SOPC Builder (voir figure 4.7).

4.4 Synthèse de haut niveau

Le prototypage rapide est considéré comme une clé pour accélérer la conception des systèmes sur du matériel [18]. Il est connu que la performance est proportionnelle à la capacité physique (ressources) du FPGA et inversement proportionnelle à la taille de la conception sur du matériel [125]. Dans ce contexte, la synthèse de haut niveau est une approche complémentaire qui recherche le meilleur compromis entre la performance et la consommation de ressources tout en réduisant le temps de conception. La synthèse de haut niveau est analogue à la compilation de logiciels transposée au domaine du matériel [48]. La synthèse de haut niveau permet la recherche (semi) automatique de solutions architecturales qui respectent les contraintes spécifiées tout en optimisant les objectifs de la conception [35].

Le principe de la synthèse de haut niveau remonte au début des années 1970, le concept ayant été proposé par M. R. Barbacci et D. P. Siewiorek [11]. Mais ce n'est qu'au debut des années 1990 que

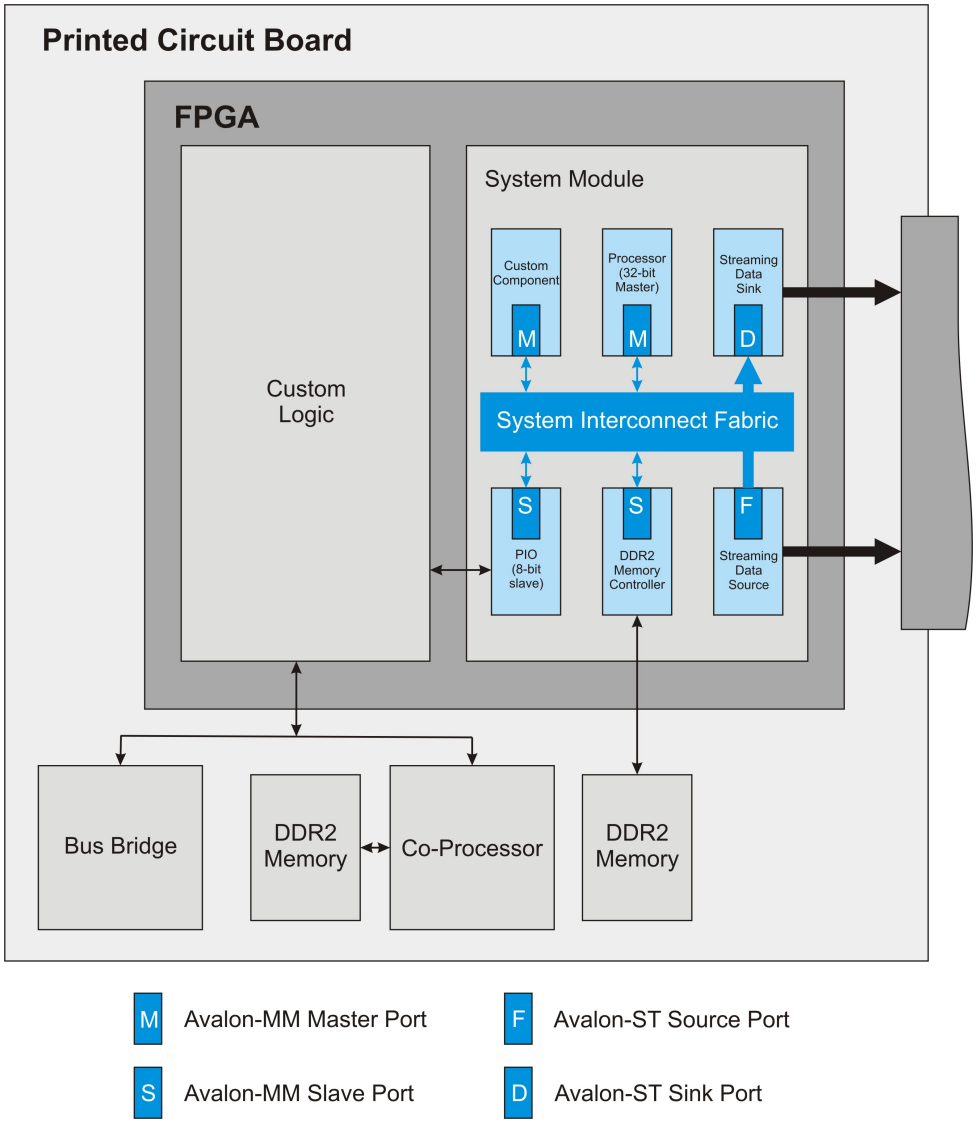


FIGURE 4.6 – Exemple de système complet sur puce généré par SOPC Builder.

les premiers produits commerciaux sont apparus : Behavioural Compiler de la société Synopsys, Alta Visual Architect de la société Cadence et Monet de la société Mentor Graphics. Toutefois, les limites technologiques liées aux performances, les ressources limitées des composants et le nombre réduit des applications ont freiné le développement des outils de synthèse de haut niveau. Au début des années 2000, la société anglaise Celoxica a popularisé le concept de compilation d'algorithmes complexes sur FPGA, en utilisant comme points d'entrée le langage Handle C. Actuellement, il existe plusieurs outils de synthèse de haut niveau, aussi académiques que commerciaux, qui ont été utilisés avec succès dans diverses applications.

Dans les paragraphes suivants nous décrirons le flot de conception sur matériel utilisant la synthèse de haut niveau puis nous présenterons quelques outils à la fois commerciaux et académiques. Nous mettrons l'accent sur l'outil GAUT (Générateur Automatique des Unités de Traitement) développé par le Lab-STICC (Laboratoire en Sciences et Technologie de l'Information de la Communication et de la Connaissance), lequel a servi à mettre en œuvre des architectures sur FPGA pour l'algorithme de

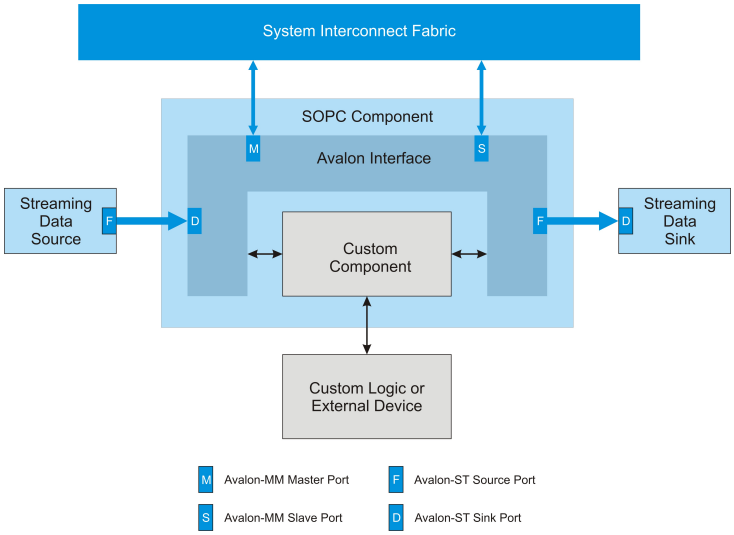


FIGURE 4.7 – Exemple d'un module de traitement personnalisé encapsulé dans un module de communication au protocole Avalon.

stéréovision dense, et nous comparerons ces architectures avec une architecture obtenue par une approche classique.

4.4.1 Flot de synthèse de haut niveau

La figure 4.8 illustre le flot de synthèse de haut niveau. Ce flot de conception a besoin de l'algorithme traduit en des spécifications fonctionnelles et des performances requises par l'application. Ce flot de conception pour architecture reconfigurable présente quatre étapes décrites brièvement ci-dessous.

Conception du modèle

Cette étape a pour but de fournir une description de l'algorithme en utilisant des composants fonctionnels. Les composants fonctionnels dépendent de l'outil de synthèse de haut niveau. Ces composants sont utilisés pour accélérer le temps de conception et l'optimisation des ressources consommées par l'architecture matérielle. Les composants fonctionnels sont contenus dans une librairie où chacun est une représentation algorithmique du composant matériel paramétré et modélisé sur l'outil de synthèse de haut niveau. Autrement dit, chaque composant fonctionnel a son composant équivalent logique sous forme d' IP. Le composant fonctionnel est utilisé au cours de la simulation et le composant matériel équivalent est utilisé dans la synthèse.

Dans la synthèse de haut niveau, théoriquement, la conception du modèle algorithmique est faite sans tenir compte de la mise en œuvre sur du matériel. Il n'est donc pas nécessaire de réaliser une adéquation algorithme-architecture au sens strict, mais en pratique c'est nécessaire lorsque les contraintes de l'application sont fortes.

Dans cette étape, il est seulement nécessaire de valider le fonctionnement de l'algorithme mis en œuvre dans l'environnement de travail de l'outil de haut niveau. Les outils de synthèse de haut niveau utilisent divers langages d'entrée comme C/C++, SystemC, MatLab, LabVIEW, etc. Un aspect important de la synthèse de haut niveau est la fiabilité du code généré et la limitation de la dépendance de la performance de l'architecture matérielle liée à l'expérience et au talent de l'ingénieur de conception.

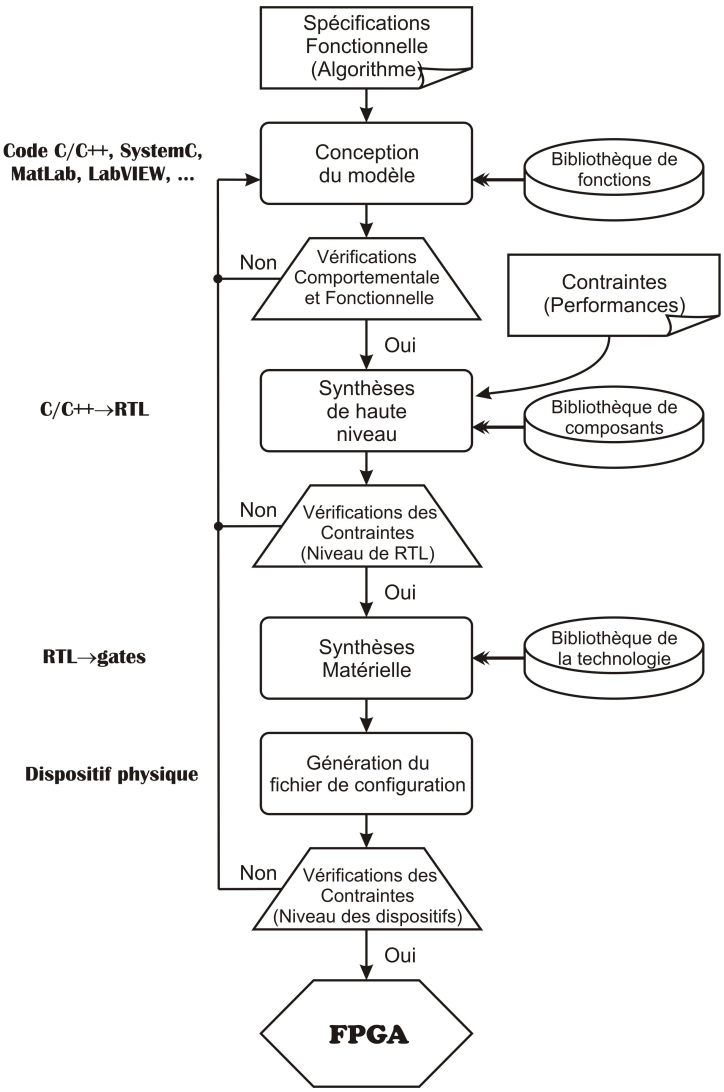


FIGURE 4.8 – Flot de synthèse de haut niveau.

Synthèse de haut niveau

La synthèse de haut niveau est la «compilation» du code dans le langage d'entrée avec un haut niveau d'abstraction vers la description architecturale au niveau RTL ou vers un fichier de configuration, prenant en compte les contraintes de l'applications. Cette approche peut être divisée généralement en quatre étapes.

1. **Compilation :** Cette étape consiste, entre autres, à supprimer le code mort, à propager les constantes, à remplacer les opérations coûteuses (multiplications et division) par des opérations plus simples (décalages, additions et soustractions) et à restructurer la représentation intermédiaire de l'algorithme à l'aide de transformations de boucles (déroulage, fusion, etc.).
2. **Ordonner et allouer les ressources matérielles :** Cette étape ordonnance et alloue les ressources matérielles nécessaires à l'exécution de l'algorithme. Les ressources sont contenues dans la bibliothèque des composants matériels. L'ordonnanceur est composé d'heuristiques qui guident le processus d'optimisation. Ces heuristiques peuvent appeler des fonctions de transformation mais

en restant toutefois indépendantes. Cela facilite le développement de nouvelles heuristiques et de nouvelles transformations.

3. **Placement et routage :** Cette étape place les opérations dans des ressources fonctionnelles, interconnecte ces ressources, place les données dans des registres et génère un contrôleur (une machine à états finis) pour la réalisation matérielle de l'ordonnancement.
4. **Génération du code RTL :** La dernière étape est la génération du code RTL, soit VHDL ou Verilog. Les ressources fonctionnelles sont implémentées sous la forme de composants, ce qui donne une structure au circuit.

Dans cette étape du flot de conception, il est nécessaire de vérifier que les contraintes de l'application sont satisfaites. Si ces contraintes ne sont pas strictes, il est alors possible de jouer avec le compromis entre la performance et la consommation des ressources, ceci afin d'optimiser l'architecture et de trouver la mise en œuvre matérielle ayant le meilleur compromis performances-ressources.

Synthèse Matérielle et Génération du fichier de configuration

Une fois générée la description de l'architecture au niveau RTL, celle-ci doit être synthétisée pour fournir le ficher de configuration du circuit programmable, lequel permettra de valider la performance et le fonctionnement du système. Certains outils de synthèse de haut niveau n'ont pas besoin de faire la synthèse matérielle car ils fournissent déjà un fichier de configuration. Le flot de conception s'en trouve réduit et le temps de conception accéléré, mais généralement ce cas de figure limite le choix des dispositifs. Enfin dans cette étape il est nécessaire de faire la dernière validation au niveau fonctionnel et au niveau des contraintes de l'application : si l'implémentation respecte toutes les exigences alors le flot de conception est terminé, sinon il est nécessaire de revenir à la conception de l'algorithme et recommencer le processus jusqu'à ce que l'implémentation réponde aux contraintes souhaitées.

4.4.2 Outils de synthèse de haut niveau

La possibilité d'accélérer le temps de conception des architectures de traitement du signal a induit, dans la dernière décennie, un grand intérêt dans la recherche de méthodes de synthèse de haut niveau lesquelles ont favorisé une croissance rapide des outils associés.

Il est possible de séparer les outils de synthèse de haut niveau en deux catégories. La première catégorie correspond aux outils principalement conçus pour porter des algorithmes de traitement sur FPGA/SOPC. Ces outils permettent une partition de l'algorithme en deux : une partie sur logiciel et une autre sur matériel. La partie logicielle est exécutée sur un processeur qui est embarqué sur le FPGA, la partie matérielle est mise en œuvre en dur et les deux sont reliées entre elles par un bus de communication. Ces outils peuvent toutefois créer une architecture matérielle capable de gérer l'algorithme complètement de manière autonome. Parmi ces outils nous pouvons cité : C2H [66] d'Altera, Impulse CoDeveloper [72] d'Impulse Acceletared Technologies et DK Design Suite [75] de Mentor Graphics.

La deuxième catégorie concerne les outils qui ont pour objectif de transformer des algorithmes, décrits dans un langage de haut niveau (C, C++, SystemC, MatLab, SynDex, LabVIEW, etc.) en une architecture matérielle intégrable dans un circuit. Cette catégorie peut être subdivisée également en deux catégories selon la représentation du langage d'entrée. La première catégorie utilise une représentation graphique de l'algorithme, la deuxième catégorie utilise une description en langages C, C++ ou SystemC.

SynDEx-IC

SynDEx-IC [87] est un outil développé en collaboration entre les laboratoires A2SI (Algorithmique et Architecture des Systemes Informatiques) du groupe ESIEE et l'INRIA Rocquencourt-OSTRE. Cet

outil utilise un algorithme spécifié sous la forme de graphe de tâches comme dans SynDex [78]. Ce graphe peut être factorisé de manière à exprimer une répétition sur une tâche, c'est à dire qu'il exprime son parallélisme de données potentiel. Les frontières de données avec les tâches factorisées sont décrites par des nœuds de factorisation Fork, Join, Diffusion et Iterate. L'heuristique utilisée dans cet outil pour l'optimisation permet de prendre en compte les contraintes temps réel de l'algorithme et les ressources disponibles pour sa mise en œuvre. L'architecture générée par cet outil satisfait généralement les contraintes de surface et de temps réel requises. Pour cela, la notion de défactorisation a été présentée par Dias et al [38]. Cette technique consiste à dérouler les boucles dans un graphe factorisé. Plus le graphe est défactorisé, plus les latences d'exécution sont faibles et plus les ressources consommées augmentent. Le but est donc de dérouler les blocs de manière à remplir les contraintes de temps d'exécution tout en essayant de minimiser les ressources requises à la mise en œuvre sur du matériel à l'issu du processus d'optimisation. Cette optimisation est intéressante car elle prend en considération les ressources disponibles sur le PFGA et les contraintes temps réel requises par l'application. Cet outil donne en sortie la spécification de l'architecture sous forme de code VHDL synthétisable sur un FPGA d'Altera ou de Xilinx.

Simulink HDL Coder et SynplifyDSP

Simulink HDL Coder [2] développé par MathWorks [74] est un outil capable de générer une description en langage matériel (VDHL ou Verilog) à partir d'un modèle décrit dans l'environnement Simulink ou Stateflow de MathWorks. Cet outil utilise l'approche de la conception basée sur des modèles pour décrire les algorithmes à mettre en œuvre sur du matériel (FPGA ou ASIC). Il est possible de générer automatiquement des banques de tests pour l'algorithme ou simuler des IP existantes. Cependant, les dépendances de données sont exprimées à l'aide d'indices, lesquels augmentent considérablement le risque d'erreurs quant à l'expression des dépendances de données sur des tableau multidimensionnels.

Synplify DSP [79] est un outil développé par Synopsys. Il utilise Simulink pour la modélisation des algorithmes, laquelle peut être faite en utilisant des blocs fonctionnels (filtres FIR, etc.) prédéfinis dans Simulink. Il est possible d'optimiser ces blocs par le biais, entre autres, de la fonction "folding" qui permet de réduire le coût de la mise en œuvre de l'architecture en contrepartie d'une perte de la puissance de calcul. La fonction folding d'un bloc correspond à la séquentialisation de l'exécution d'un calcul réalisé par ce bloc et est paramétrable par un degré de séquentialisation choisi. Cet outil permet une implémentation sur différents fabricants de FPGA comme Altera, Xilinx, etc.

LabVIEW FPGA

LabVIEW FPGA [77] est une extension de LabVIEW développé par National Instruments. Cet outil permet de créer des VIs (Virtuel Instrument) qui s'exécutent sur des cartes FPGA du même fabricant et quelques uns de Xilinx. Bien que LabVIEW permette de manière générale d'exprimer les parallélismes et les flux de données, les VIs de LabVIEW FPGA restent limités par rapport aux VIs de LabVIEW, car toutes les fonctions ne peuvent pas être mises en œuvre sur FPGA, notamment celles liées au traitement multidimensionnel.

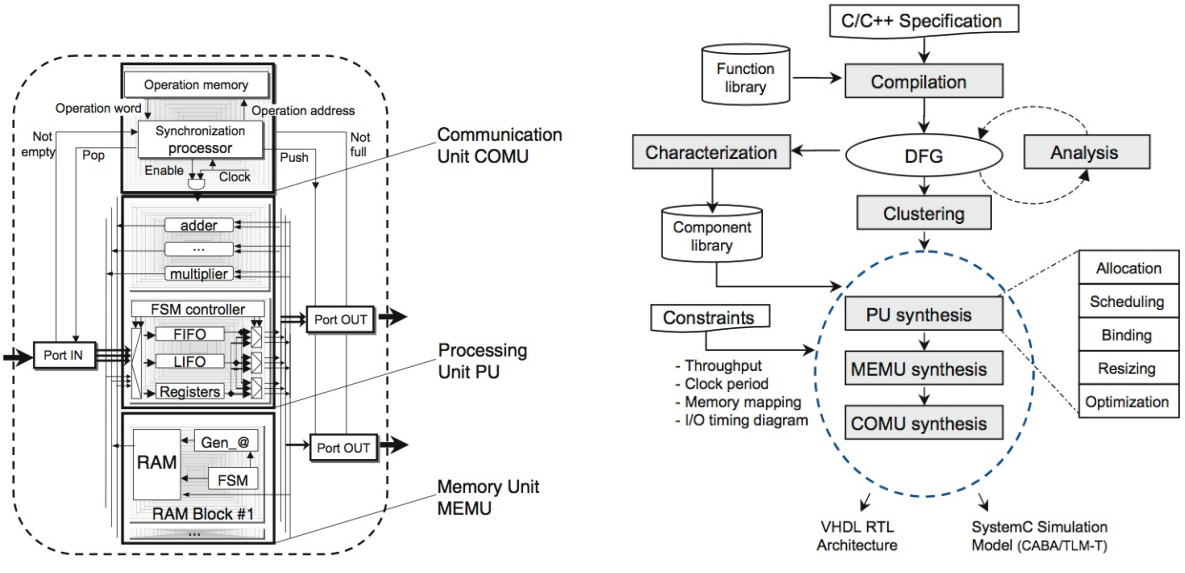
Langages C, C++ ou SystemC

Dans la dernière décennie, l'une des plus fortes tendances dans le domaine de la synthèse de haut niveau est la description des algorithmes dans le langage C, C++, SystemC ou « à la C». Le choix de ce langage est généralement lié aux habitudes des programmeurs et il existe de nombreux outils, aussi commerciaux qu'académiques. Côté commercial, on a vu l'arrivée des logiciels C-to-Hardware Compilation [67] pour Altium, AutoPilot [68] pour AutoESL Design Technologies, C-to-Silicon Compiler [69]

pour Cadence Design Systems, C2R Compiler [70] pour CebaTech, CatapultC Synthesis [75] pour Mentor Graphics, CyberWorkBench [76] pour NEC ou Synphony C Compiler [79] pour Synopsys. Côté académique, on peut citer les outils SPARK [54] de l'université de San Diego, Stream-C [47] du laboratoire national de Los Alamos, SA-C [121] de l'université du Colorado, ROCCC [63, 113, 53] de l'université de Riverside ou GAUT [65, 104] du Lab-STICC en France. Il existe encore de nombreux outils et notre intention n'est pas de les décrire tous. Nous détaillerons ci-dessous l'outil GAUT que nous avons choisi de tester dans le cadre d'une de nos applications.

Générateur Automatique des Unités de Traitement

Le Générateur Automatique des Unités de Traitement (GAUT [65]) est un outil de synthèse de haut niveau universitaire et open-source consacré aux applications de traitement numérique de signaux et d'images sous contraintes d'exécution en temps réel [104]. GAUT prend en entrée une spécification algorithmique en C/C++ qui doit être synthétisée. Les contraintes obligatoires sont le débit et la période d'horloge, tandis que le mappage de la mémoire et le diagramme de temps des entrées et sorties sont des contraintes facultatives de la conception. La figure 4.9a montre l'architecture des composants matériels que génère GAUT. Cette architecture est composée de trois unités fonctionnelles principales : une unité de traitement (PU), une unité de mémoire (MEMU) et une unité de communication et interface (COMU). La PU est un datapath composé d'opérateurs logiques et arithmétiques, d'éléments de stockage, de logique de pilotage et d'un contrôleur (FSM). Les éléments de stockage de la PU peuvent être des mémoires sémantiques (FIFO, LIFO) et/ou des registres. La MEMU est composée de banques de mémoires et de leurs contrôleurs correspondants. La COMU comprend un processeur de synchronisation et une mémoire d'exploitation qui permet d'avoir une interface de communication GALS/LIS (Globally Asynchronous Locally Synchronous/Latency Insensitive System) [35].



(a) (b)

FIGURE 4.9 – (a) L'architecture matérielle générée par GAUT, (b) Le flot proposé de synthèse de haut niveau.

La figure 4.9b décrit le flot de synthèse de haut niveau dans lequel on peut voir que GAUT synthétise l'unité de traitement en premier, puis l'unité de mémoire suivie de l'unité de communication. Durant

la conception de la PU, GAUT sélectionne d'abord les opérateurs arithmétiques et après optimise leur utilisation dans l'architecture en fonction des contraintes et des objectifs de la conception. Puis GAUT traite les registres et les banques de mémoire qui font partie de l'unité de mémoire. L'optimisation des registres est faite avant l'optimisation de la mémoire laquelle est basée sur des techniques de prédiction. Les interconnexions sont ensuite optimisées suivies par l'optimisation des générateurs d'adresses des banques de mémoires dédiées à l'application considérée. L'interface de communications est ensuite générée en utilisant le comportement temporel des entrées et des sorties dans le composant [35].

Un fichier de test est généré automatiquement pour valider l'architecture produite. Ces tests consistent à appliquer des stimulis à l'architecture et à analyser les résultats. Les stimulis peuvent être incrémentaux, aléatoires ou des valeurs définies par l'utilisateur, permettant ainsi de faire une comparaison automatique avec le cahier de charges initial de l'algorithme (modèle «idéal»). Il est possible de vérifier seulement l'unité de traitement. Dans ce cas, les unités de mémoire et de communication sont générées en tant que composants VHDL dont le comportement est décrit comme une machine à états finis avec interconnexions. GAUT génère non seulement les modèles VHDL mais aussi les scripts nécessaires pour compiler et simuler l'architecture avec ModelSim. Il peut également comparer les résultats de deux simulations (produites par des comportements de temps (entrées/sorties, pipeline, etc.). Les deux modèles de simulation «Cycle Accurate, Bit Accurate» (CABA) et «Transaction-Level Model with Timing» (TLM-T) sont générés, permettant d'intégrer ces composants dans la plateforme SoCLib [65]. GAUT aborde également la conception des architectures multi-modes [20].

4.5 Analyse comparative : Stéréovision passive dense

Dans cette section nous effectuons une analyse comparative de deux flots de conception pour l'algorithme de stéréovision dense. Le flot de conception classique jusqu'au niveau RTL nous a fourni une architecture de base pour la comparaison et est évalué en premier. Enfin le même algorithme a été mis en œuvre en utilisant l'outil de synthèse de haut niveau GAUT. Les deux architectures obtenues ont été évaluées pour faire une comparaison des deux approches.

Dans cette section nous donnons une brève description de l'algorithme de stéréovision dense puis nous expliquons la mise en œuvre de l'architecture en utilisant le flot de conception au niveau RTL. Ensuite, l'architecture correspondante obtenue par synthèse de haute niveau est expliquée également. Pour conclure une analyse comparative des deux architectures obtenues et de quatre architectures issues de la littérature résolvant le même problème est effectuée.

4.5.1 Une vue d'ensemble de l'algorithme de stéréovision dense.

Dans la vision par ordinateur, la stéréovision a pour but de récupérer les informations de profondeur à partir de deux images issues de la même scène. Un pixel dans une image correspond à un pixel dans l'autre, si les deux pixels sont projetés depuis la vue du même élément physique dans la scène. Aussi, si les deux images sont séparées mais simultanées, alors le calcul de la correspondance détermine la profondeur stéréo [143]. Il y a deux approches principales pour traiter la corrélation stéréo : les approches basées sur les caractéristiques et par zones. Dans ce travail, nous nous sommes plutôt intéressés à l'approche par zone, car elle propose une solution dense qui produit des images de disparité de haute densité. De plus, elle a une structure algorithmique très régulière bien adaptée pour le portage sur une architecture matérielle. La transformation census est l'approche par zone choisie pour résoudre le problème de la stéréovision.

L'algorithme global de la stéréovision dense basé sur la transformation census est présenté dans la figure 4.10. Tout d'abord les images gauche et droite sont traitées indépendamment. Afin de diminuer

la complexité du matériel et le nombre d'opérations au cours du traitement des images, la restriction épipolaire est utilisée. En raison de cette restriction, liée au positionnement géométrique des deux caméras qui voient leurs axes principaux parallèles, les lignes épipolaires se situent le long des lignes ou des colonnes. Ainsi la différence de position des caméras se réduit à une translation horizontale ou verticale et tout pixel visible dans une ligne épipolaire d'une image apparaît dans une position décalée dans la ligne épipolaire correspondante de l'autre image, ceci en supposant que le pixel n'est pas caché dans la seconde image [7].

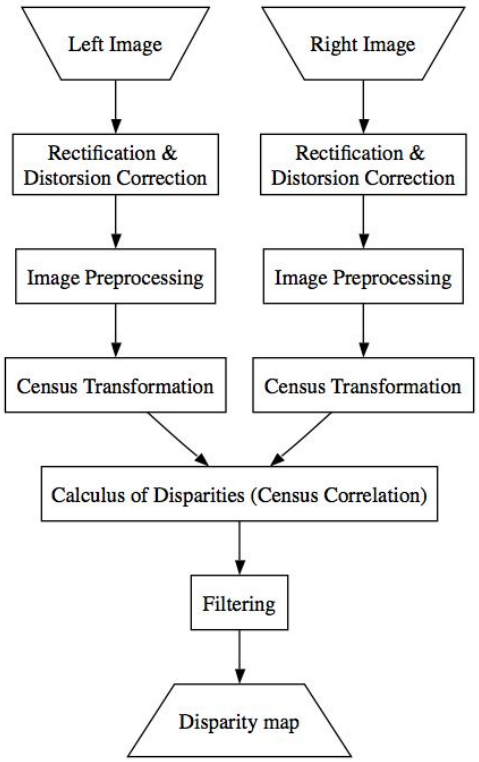


FIGURE 4.10 – Algorithme de stéréovision dense basé sur la transformation census.

Avant de calculer la transformation census, un prétraitement des deux images est exécuté, lequel se compose d'un filtre de moyenne arithmétique. Soit S_{uv} l'ensemble des coordonnées dans une fenêtre rectangulaire de taille $m \times n$ pixels et qui est centrée sur le point (u, v). Le processus de filtrage par la moyenne arithmétique calcule la valeur moyenne de l'image bruitée ou dégradée $I(u, v)$ dans la zone définie par S_{uv}. La valeur de l'image restituée $\hat{I}$ en tout point (u, v) est simplement la moyenne arithmétique calculée à l'aide de tous les pixels de la région définie par S_{uv}. La formule de la moyenne arithmétique utilisée est

$$\hat{I}(u, v) = \frac{1}{mn} \sum_{(i,j) \in S_{uv}} I(i, j) \tag{4.1}$$

Cette opération peut être réalisée sans utiliser le facteur d'échelle $1/mn$, seulement si le rang correct de l'opération est considéré. Le filtre de moyenne simple lisse les variations locales dans une image. Egalement dans cette opération, le bruit lié au flou est réduit.

Ensuite la transformation census des images prétraitées est calculée. La mesure non paramétrique connue sous le nom de transformation census a été introduite par Zabih et Woodfill [143]. Cette transfor-

mation est calculée dans un voisinage de pixels et basée sur les relations entre eux, comme illustré dans l'équation 4.2.

$$I_C(u,v) = \bigotimes_{(i,j)\in D_{uv}} \xi\left(\hat{I}(u,v), \hat{I}(i,j)\right) \tag{4.2}$$

où D_{uv} représente l'ensemble des coordonnées dans une fenêtre carrée de taille $n \times n$ pixels, où n est un nombre impair, et qui est centrée sur le point (u,v). La fonction ξ calcule la relation entre un pixel (u,v) et ses plus proches voisins dans D_{uv}. Cette fonction retourne un bit, qui est mis à un, lorsque l'intensité du point de (i,j) est inférieure à l'intensité du point (u,v), sinon le bit est mis à zéro. L'opérateur $\otimes$ désigne l'opération de concaténation. Enfin, I_C représente la transformation census du point (u,v), qui est une chaîne de bits.

Une fois obtenues les images de la transformation census, deux pixels (un pixel par image) sont comparés par similitude avec la distance de Hamming pour obtenir la mesure de disparité. Ainsi, l'évaluation de similarité est basée sur la comparaison binaire entre deux chaînes de bits données par la transformation census. La mesure des disparités D_H dans le point (u,v) est calculée par l'équation 4.3, où I_{Cl} et I_{Cr} représentent les deux images de la transformation census, gauche et droite respectivement. La disparité est le résultat de la maximisation de similarité entre deux images pour la même ligne épipolaire v. Dans la même équation, D représente la valeur maximale de déplacement de la ligne épipolaire de l'image de droite. Enfin, $\bar{\otimes}$ désigne l'opération binaire XNOR.

$$D_H(u,v) = \max_{d\in[0,D]} \left(\frac{1}{N} \sum_{i=1}^{N} I_{Cl}(u,v)_i \,\bar{\otimes} I_{Cr}(u-d,v)_i \right) \tag{4.3}$$

Afin d'améliorer la qualité de l'image de disparité, un processus de filtrage est nécessaire. Ce processus se compose d'un filtre spatial de médian. Soit M_{uv} l'ensemble des coordonnées dans une fenêtre rectangulaire de taille $m \times n$ pixels et qui est centrée sur le point (u,v). Le processus de filtrage par la médiane calcule la valeur placée au milieu de la suite ordonnée de l'image de disparité $D_H(u,v)$ dans la zone définie par M_{uv}. La valeur de l'image filtrée $\tilde{D}_H$ en tout pixel (u,v) est simplement la médiane calculée sur tous les pixels de la région définie par M_{uv}. L'équation 4.4 montre la formule du filtre de médian.

$$\tilde{D}_H(u,v) = \text{median}\left(D_H(i,j), (i,j) \in M_{uv}\right) \tag{4.4}$$

4.5.2 Mise en œuvre matérielle utilisant le flot de conception au niveau RTL

Notre architecture a été développée pour une cible FPGA et conçue comme un moteur de traitement d'image général à haute vitesse. Afin de maximiser la performance, la conception a été guidée par une minimisation des ressources consommées par le FPGA. Comme nous l'avons expliqué dans la section précédente, l'architecture est composée de trois opérations principales : le prétraitement de l'image, la transformation census et la corrélation census. La mise en œuvre sur matériel de ces processus est détaillée dans les paragraphes suivants.

Le prétraitement : le filtre de moyenne arithmétique

L'image acquise par la caméra n'étant pas très bruitée, une fonction de lissage rapide peut être utilisée. Ainsi, la moyenne arithmétique est réalisée sur une fenêtre de taille de 3×3 pixels ce qui est suffisant pour poursuivre l'algorithme et permet une économie de ressources notable pour l'architecture finale. Le calcul de la moyenne est réalisé en deux étapes : les additions horizontales puis verticales.

L'architecture correspondant à ce module est représentée dans la figure 4.11. Les trois registres sont utilisés pour réaliser l'addition horizontale. Ces registres sont reliés à deux additionneurs parallèles de huit bits, le résultat étant codé sur dix bits. Le résultat de l'addition horizontale est stocké dans une mémoire deux fois plus grande que la taille de la largeur de l'image. L'addition verticale est calculée en prenant le résultat de l'addition horizontale courante ainsi que les résultats des additions horizontales des deux lignes précédentes stockées en mémoire. La moyenne arithmétique des neuf pixels de la fenêtre est codée sur douze bits. A ce stade, la latence ne dépend que du stockage de la dernière ligne plus un pixel.

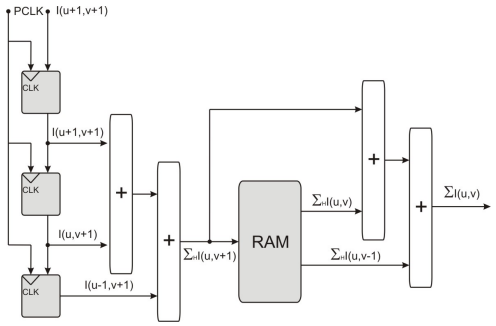


FIGURE 4.11 – Architecture du module de calcul de la moyenne arithmétique.

La transformation census

Les moyennes arithmétiques des images gauche et droite, obtenues à l'étape précédente, sont utilisées comme entrées pour l'étape de transformation census. Cette transformation encode toutes les valeurs d'intensité contenues dans une fenêtre de recherche par rapport à la valeur d'intensité centrale. Le module correspondant à cette architecture est représenté dans la figure 4.12. La performance de la corrélation census dépend de la taille de la fenêtre de recherche utilisée. Cette taille de fenêtre ayant une incidence considérable sur les ressources consommées et le temps de traitement, il y a lieu de la choisir de manière optimale. Par conséquent, après plusieurs tests, nous avons sélectionné une taille de fenêtre de recherche de 7×7 pixels, car elle présente des résultats convenables pour un bon compromis entre le temps de traitement et la taille du matériel. Cette fenêtre nécessite 49 registres. Le module de traitement est également composé de six blocs mémoire, dont la taille est obtenue en soustrayant la largeur de la fenêtre de recherche (7 pixels dans notre cas) à la largeur de l'image (640 pixels). Ce résultat est multiplié par 12 bits afin que la taille du bus d'entrée de la transformation census soit égale à la taille du bus de sortie de la moyenne arithmétique. Une fois la structure de la fenêtre de recherche mise en place, la procédure de la transformation census est exécutée. Pour ce faire, le pixel central de la fenêtre de recherche est comparé avec ses 48 voisins locaux. Cette dernière opération exige que tous les registres correspondants soient connectés à des comparateurs en parallèle (comme il est montré dans la figure 4.12). Enfin, le résultat est codé sur 48 bits, chaque bit correspondant à la sortie d'un comparateur. A ce stade, la latence ne dépend que de la moitié de la taille de la fenêtre de recherche multipliée par la largeur de l'image.

La corrélation census

La tâche de corrélation consiste à analyser les images de transformation census droite et gauche en tenant compte du fait que celles-ci contiennent des objets communs. Elle sert à trouver le déplacement entre deux pixels projetés depuis un même objet pris à partir de deux points de vue différents correspondant aux images droite et gauche ; ce déplacement est appelé mesure de disparité. Dans la plupart des cas, la corrélation utilise la maximisation de similarité entre deux images afin de trouver la disparité ;

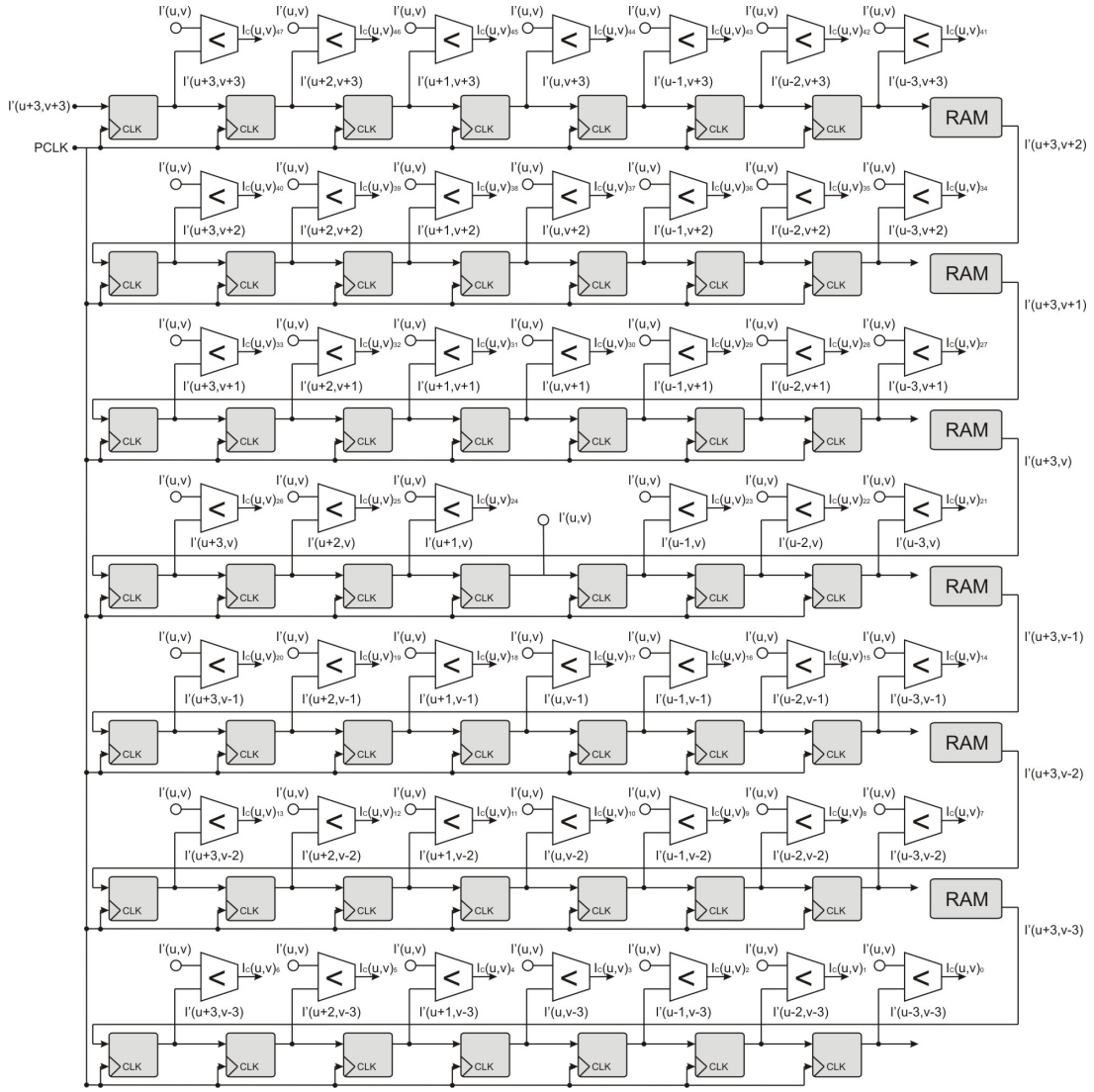


FIGURE 4.12 – Architecture du module de calcul de la transformation census.

c'est la méthode que nous avons retenue pour cette phase. La procédure comprend deux étapes principales : d'une part, calculer la similarité, d'autre part, rechercher la valeur maximale. La figure 4.13 montre l'architecture du module correspondant. Afin de réduire la latence lors de la tâche de corrélation, il est plus approprié de comparer un point dans l'image de la transformation census gauche avec le nombre maximum de points dans l'image de la transformation census droite correspondant. Dans notre application, le nombre maximum de points dans l'image de la transformation census droite a été établi à 64 pixels, lesquels sont stockés dans des registres représentés sous forme de blocs gris sur la gauche de la figure 4.13. Les registres et le pixel de la transformation census gauche sont les entrées des opérateurs binaires $XNOR$ qui fournissent un chaîne de 48 bits en sortie. Dans la deuxième partie de la tâche, chaque bit est sommé dans le but de trouver la similitude entre le pixel de la transformation census gauche et tous les pixels de la transformation census de droite. Si un pixel dans la ligne de l'image de la transformation census droite est le même que le pixel de l'image de la transformation census gauche correspondant, les bits résultant de l'opération $XNOR$ seront tous mis à un, et par conséquent la similitude sera maximale. Au contraire, si tous les pixels comparés sont différents, l'opération $XNOR$ retournera zéro, de sorte

que la similitude sera mise à zéro.

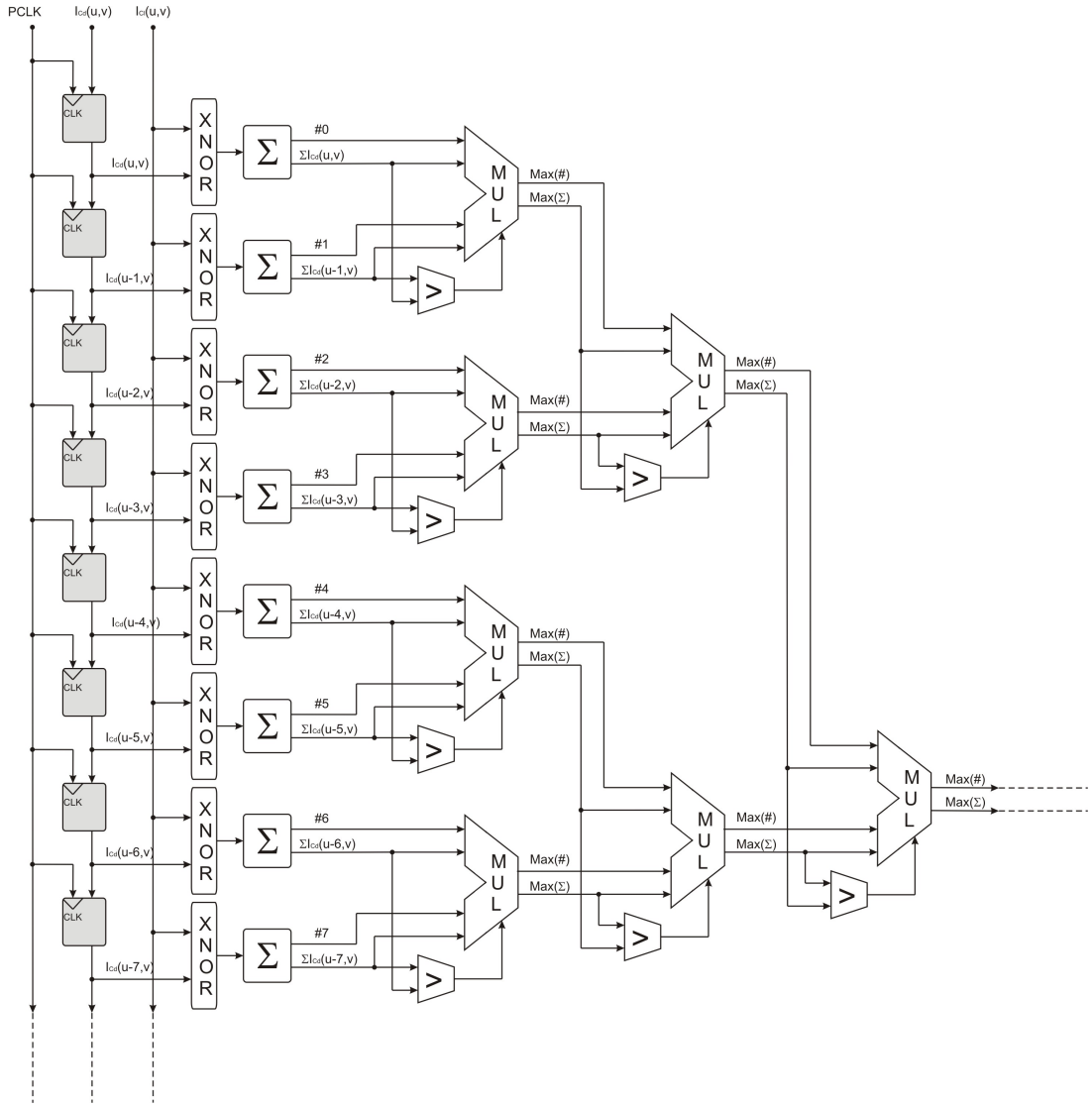


FIGURE 4.13 – Architecture du module de calcul de la corrélation census.

Une fois que le calcul de la similarité a été obtenu, la maximisation de la similarité suit, ce qui signifie une recherche de la valeur maximale parmi les 64 pixels comparés. Cette étape comprend plusieurs unités de comparaison constituées de quatre entrées chacune : deux valeurs de similarité ou scores et leurs indices correspondant. Ces indices représentent le déplacement entre les pixels de la transformation census gauche et droite. Autrement dit, si un pixel de la transformation census droite a la même position que le pixel de la transformation census gauche, l'indice correspondant est égal à zéro, sinon, l'indice représente le nombre de pixels déplacés entre la position du pixel de la transformation census gauche et la position du pixel de la transformation census droite. Selon l'architecture du module représenté dans la figure 4.13, nous pouvons dire que le processus de maximisation de la similarité est mis en œuvre dans une unité de sélection qui reçoit en entrée 2 mesures de similarité et les 2 índices correspondant. Les composants internes de l'unité de sélection sont un multiplexeur, qui choisit l'índice du pixel ayant la plus forte similarité, et un comparateur. Ce dernier prend en entrée deux valeurs de similarité issues de la première étape ; la sortie du comparateur sert à contrôler le multiplexeur. Ce multiplexeur donne en sortie

la mesure de similarité et l'indice du pixel correspondant. Enfin, afin d'obtenir le pixel avec la mesure de similarité maximale, six couches d'unités de sélection sont nécessaires. Si nous représentons ces couches sous la forme d'une pyramide de plusieurs niveaux, le niveau le plus bas (qui correspond à notre première couche), exécute 32 fois la procédure d'une unité de sélection décrite ci-dessus. Comme nous nous déplaçons vers le haut de la pyramide, chaque niveau réduit de moitié le nombre d'opérations du niveau précédent ce qui nous donne six niveaux au total. Le dernier niveau donne la mesure de similarité ou score maximal et la disparité entre les pixels correspondants dans les images census droite et gauche. A ce stade, la latence dépend du nombre de couches d'unités de sélection lequel est fonction de la valeur maximale de la disparité (64 pixels dans notre cas soit 6 couches).

Dans certains cas, un processus de filtrage est nécessaire pour améliorer la qualité de la carte de disparité. Pour minimiser la complexité de l'architecture, nous avons évité ce processus. Ainsi, la carte obtenue est directement issue de la corrélation census.

Résultats de synthèse

Nous avons fait une première mise en œuvre de l'algorithme de stéréovision dense basée sur la transformation census sur du matériel en utilisant un flot de conception au niveau RTL [82]. L'architecture a été décrite en langage VHDL dans les environnements Quartus II et ModelSim puis synthétisée pour un composant cible EP2C35F672C6 de la famille Cyclone II d'Altera [66].

La synthèse sur FPGA donne les résultats suivants : l'architecture mise en œuvre exige $11,683$ fonctions combinatoires et $5,162$ registres logiques dédiés, qui représentent tous le deux un total de $12,183$ éléments logiques. La mémoire nécessaire est de $112,025$ bits. La quantité d'éléments logiques ne représente que les 37% de la capacité totale du circuit tandis que la taille de la mémoire représente 43%. Les ressources requises par cette architecture sont principalement fonction de la taille de l'image, la taille des fenêtres de recherche utilisées par le filtre de moyenne et la transformation census et la valeur maximale de la mesure de disparité. Dans cette architecture, nous utilisons une taille d'image de 640×480 pixels, des tailles de fenêtres de recherche de 3×3 et 7×7 pixels pour le filtre de moyenne et la transformation census respectivement et une disparité maximale de 64 pixels. Avec ces paramètres, l'architecture est capable de calculer 130 images de disparité par seconde avec un rythme d'horloge de 50 Mhz.

4.5.3 Exploration architecturale mettant en oeuvre la synthèse de haut niveau

Nous avons fait une exploration architecturale de l'algorithme de stéréovision dense basé sur la transformation census en utilisant le flot de synthèse de haut niveau [84]. L'algorithme a été développé sur l'outil GAUT puis synthétisé pour le même circuit (EP2C35F672C6). Chaque étape de l'algorithme a fait l'objet de plusieurs architectures résultant de différents compromis entre la performance (mesurée avec le débit ou la cadence) et les ressources consommées.

Dans les tables 4.1 à 4.3 nous présentons les trois architectures repérées Design 1, 2 et 3 avec les performances les plus représentatives. Afin de mieux interpréter les informations présentées dans ces tables et savoir comment l'architecture finale a été choisie, nous allons décrire la relation entre les caractéristiques afin d'obtenir le meilleur compromis entre les ressources nécessaires et la performance. Si la performance est réduite, puis augmente la cadence, par conséquence, le nombre des opérateurs et des étapes dans le pipeline baisse. Les autres caractéristiques suit des relations moins évidentes. C'est, le nombre d'éléments logiques sont une conséquence directe des deux : les fonctions combinatoires et le numéro de registres logiques dédiés. Alors que les fonctions combinatoires dépendent fortement du nombre d'opérateurs et faiblement du nombre d'états de la machine à états qui dépend donc du temps de cadence. Contrairement, les registres logiques dédiés dépendent fortement du nombre d'états de la

machine à états et faiblement du nombre d'opérateurs. Enfin, la latence dépend du nombre d'opérateurs, le nombre d'étapes dans le pipeline et d'une manière plus forte du temps de cadence en raison de la conception de l'architecture. Les résultats montrés dans les tables ont été effectués en utilisant une taille d'image de 640×480 pixels, avec une taille de fenêtre de recherche de 3×3 et 7×7 pixels pour le filtre de moyenne et la transformation census, respectivement, et une mesure de disparité maximale de 64 pixels, pour une rythme d'horloge de 100 Mhz.

TABLE 4.1 – Table comparative pour le filtre de moyenne arithmétique.

Caractéristiques	Design 1	Design 2	Design 3
Cadence (ns)	20	30	40
Performance (fps)	160	100	80
Eléments Logiques	118	120	73
Fonctions Combinatoires	86	72	73
Registres Logiques Dédiés	115	116	69
# Etapes dans le pipeline	3	2	2
# Opérateurs	2	2	1
Latence (μs)	25.69	38.52	51.35

TABLE 4.2 – Table comparative pour la transformation census.

Caractéristiques	Design 1	Design 2	Design 3
Cadence (ns)	40	80	200
Performance (fps)	80	40	15
Eléments Logiques	2,623	1,532	1,540
Fonctions Combinatoires	2,321	837	864
Registres Logiques Dédiés	2,343	1,279	1,380
# Etapes dans le pipeline	48	24	10
# Opérateurs	155	79	34
Latence (μs)	154.36	308.00	769.50

Compte tenu de nos contraintes nous avons choisi le design 3 pour implémenter le module du filtre de moyenne car il présente le meilleur compromis entre ressources consommées et performance. Pour les mêmes raisons le design 2 a été choisi pour développer la transformation census et le design 3 pour la corrélation census. Les résultats de la synthèse matérielle sur FPGA sont résumés comme suit : l'architecture globale mise en œuvre a besoin de $6,977$ éléments logiques et de 112.025 bits de mémoire. La quantité d'éléments logiques ne représente que 21% de la capacité totale du circuit tandis que la taille de la mémoire représente 23%. Cette architecture est capable de calculer 40 images de disparité par seconde à un rythme d'horloge de 100 Mhz.

4.5.4 Analyse comparative des architectures

Tout d'abord, nous allons analyser les performances de quatre solutions différentes pour obtenir des images de disparité : deux sont les implémentations sur du matériel présentées précédemment, une

TABLE 4.3 – Table comparative pour la corrélation census.

Caractéristiques	Design 1	Design 2	Design 3
Cadence (ns)	20	40	80
Performance (fps)	160	80	40
Eléments Logiques	1,693	2,079	2,644
Fonctions Combinatoires	1,661	1,972	2,553
Registres Logiques Dédiés	1,369	1,451	1,866
# Etapes dans le pipeline	27	12	8
# Opérateurs	140	76	46
Latence (ns)	290	160	160

solution sur un DSP pour un processeur ADSP-21161N (à un rythme d'horloge de 100 Mhz) d'Analog Devices et la dernière est testée sur logiciel pour PC DELL Optiplex 755 avec un processeur Inter Core 2 Dou à 2.00 GHz et 2 Gb de mémoire RAM. La comparaison des performances entre ces solutions est présentée dans la table 4.4. La première colonne indique les différentes tailles d'images utilisées dans les tests, de même, la deuxième colonne présente les différentes tailles de fenêtre de recherche pour la transformation census. Dans la troisième colonne, le temps de traitement (la performance) à chaque implémentation est montré. Dans le cas d'implémentations sur FPGA, le traitement en parallèle est mis à profit pour obtenir des temps de calcul très petits. L'architecture développée en utilisant la conception au niveau RTL a le meilleur temps de traitement mais prend le plus de temps de conception. Inversement l'architecture développée en utilisant la synthèse de haut niveau est moins complexe, a un temps de traitement convenable et un temps de conception beaucoup plus court. Contrairement aux implémentations sur FPGA, la solution sur DSP est plus rapide et plus facile à mettre en œuvre, mais le traitement restant majoritairement séquentiel, les temps de calcul s'en trouvent augmentés. Enfin, la mise en œuvre la plus souple est la solution PC, mais les temps de traitement sont sans comparaison avec les solutions matérielles et inappropriées pour les applications temps réel.

TABLE 4.4 – Table comparative des performances pour différentes implémentations.

Taille de l'image (pixels)	Taille de la fenêtre census (pixels)	Temps de traitement (Performance) FPGA (RTL)	FPGA (HLS)	DSP	PC
192×144	3×3	$0.69ms$	$2.25ms$	$0.26s$	$33.29s$
192×144	5×5	$0.69ms$	$2.25ms$	$0.69s$	$34.87s$
192×144	7×7	$0.69ms$	$2.25ms$	$1.80s$	$36.31s$
384×288	3×3	$2.77ms$	$9.00ms$	$1.00s$	$145.91s$
384×288	5×5	$2.77ms$	$9.00ms$	$2.75s$	$151.39s$
384×288	7×7	$2.77ms$	$9.00ms$	$7.20s$	$158.20s$
640×480	3×3	$7.68ms$	$25.00ms$	$2.80s$	$403.47s$
640×480	5×5	$7.68ms$	$25.00ms$	$7.70s$	$423.63s$
640×480	7×7	$7.68ms$	$25.00ms$	$20.00s$	$439.06s$

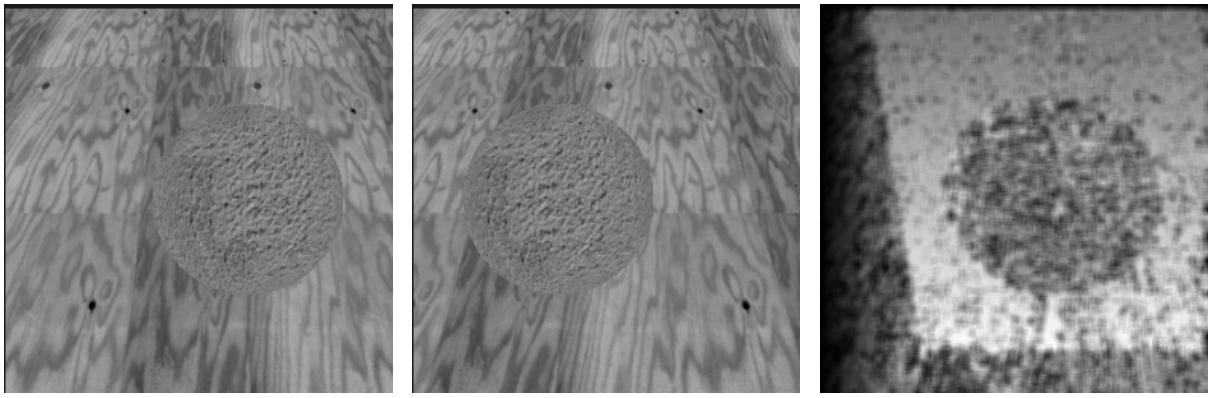
Nous présentons une analyse comparative de nos deux architectures et quatre implémentations sur FPGA trouvées dans la littérature. Les caractéristiques les plus importantes sont présentées dans la première colonne de la table 4.5. La deuxième et la troisième colonnes montrent les contraintes, les performances et les ressources consommées par nos deux architectures développées en utilisant la conception au niveau RTL et la synthèse de haut niveau HLS, appelées respectivement Design 1 et Design 2. Les autres colonnes présentent les valeurs correspondantes de quatre architectures, étiquetées Design 3 à 6, mises en œuvre par différents auteurs (pour plus de détails techniques voir les références [114], [111], [7] et [110], respectivement). Elles sont toutes implémentées sur FPGA et calculent une carte de disparité à partir de deux images stéréo. Nos architectures peuvent être comparées directement avec les Designs 3 et 4 car elles utilisent l'algorithme de la transformation census pour calculer la carte de disparité. Pour l'architecture Design 1, nous obtenons deux améliorations importantes par rapport au Design 3 : ce sont les temps de latence et la taille de la mémoire. Le coût de ces améliorations se reflète dans le nombre d'éléments logiques, lequel est plus important dans notre cas. Pour l'architecture Design 2, nous obtenons trois améliorations importantes par rapport au Design 3, ce sont le temps de latence, le nombre d'éléments logiques et la taille de la mémoire. Le coût de ces améliorations se reflète dans la performance en terme d'images par seconde qui est plus faible. Bien que le Design 4 ait une performance en terme d'images par seconde similaire il utilise une image quatre fois plus petite, une disparité maximale inférieure et davantage d'éléments logiques et de capacité mémoire. Il est inapproprié de comparer les Designs 5 et 6 car ils utilisent les différences de somme absolue (SAD) en tant que mesure de corrélation. Cependant le point de comparaison intéressant réside dans la performance des architectures pour calculer la carte de disparité lorsque celle-ci utilise seulement des éléments logiques (Design 5) ou quand elle fait de nombreux accès à la mémoire externe (Design 6). C'est le nombre d'éléments logiques dans le Design 5 qui limite à la fois la taille de l'image et la valeur maximale de la disparité, par conséquent cette architecture a une plus faible performance que celle du Design 1. Le Design 6 utilise, quant à lui, trop les mémoires externes et par conséquent la performance de l'architecture se trouve sévèrement réduite par rapport aux deux nôtres.

TABLE 4.5 – Table comparative de la performance pour différentes architectures sur FPGA.

	Design 1 RTL [82]	Design 2 HLS [84]	Design 3 Naoulou [114]	Design 4 Murphy [111]	Design 5 Arias [7]	Design 6 Miyajima [110]
Measure	Census	Census	Census	Census	SAD	SAD
T. d'image	640×480	640×480	640×480	320×240	320×240	640×480
T. de fenêtre	7×7	7×7	7×7	13×13	7×7	7×7
Max. disparité	64	64	64	20	16	80
Performance	130	40	130	40	71	18.9
Latence (μs)	115	206	274	–	–	–
Surface	$12,188$	$6,977$	$11,100$	$26,265$	$4,210$	$7,096$
T. de mémoire	114 Kb	109 Kb	174 Kb	375 Kb	–	–

Les deux architectures de stéréovision (RTL et HLS) basées sur la transformation census ont été testées sur trois paires d'images stéréo. Celles-ci ayant donné les mêmes résultats de traitement, nous ne montrerons et n'expliquerons qu'une seule des images de disparités. Les figures 4.14a et4.14b montrent les images gauche et droite de la première paire d'images stéréo. A partir de cette paire d'images de synthèse, on peut voir qu'il y a un grand écart entre elles : l'image gauche montre la texture de fond dans la même direction que l'axe central alors que l'image droite présente un peu d'inclinaison. L'image

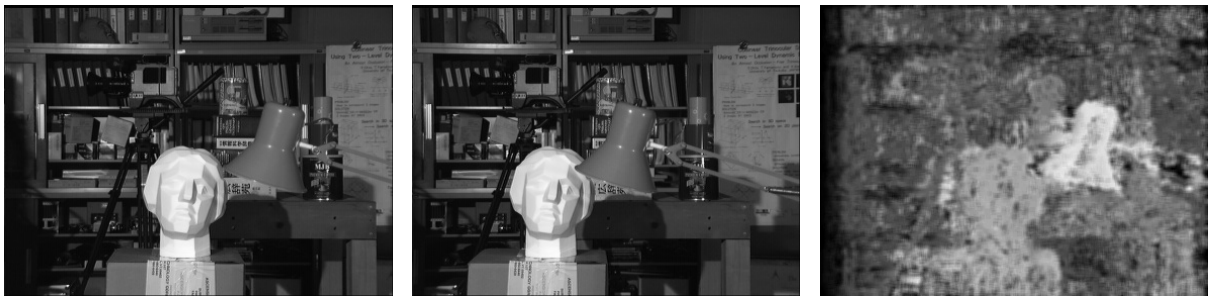
de disparité résultant de l'implémentation sur FPGA (illustrée sur la figure 4.14c) montre la plupart des points qui forment le cercle, dans un même plan (un seul niveau de gris dans le cercle) et l'arrière-plan dans différents niveaux de gris. Pour expliquer les résultats correspondant à l'arrière-plan, nous devons rappeler que le processus de corrélation est effectué de gauche à droite dans l'image, ainsi les positions plus semblables sur l'image résultant sont les positions sur l'image gauche. En outre, les pixels noirs sur la gauche de la figure 4.14c sont relatifs à des points de l'image gauche qui n'ont pas de correspondant dans l'image droite.



(a) (b) (c)

FIGURE 4.14 – Les images stéréo et la carte de disparité, a) et b) Une paire d'images de synthèse stéréo, c) Simulation de l'implémentation sur FPGA

La figure 4.15 montre les images stéréo bien connues de l'Université de Tsukuba que nous utilisons pour tester nos architectures. Ces images présentent plusieurs plans de profondeur. L'image de disparité résultante est montrée dans la figure 4.15c, comme dans le cas précédent, les pixels sombres à gauche correspondent aux pixels introuvables. Malgré des conditions de luminosité contrôlée, cette image permet une évaluation intéressante des performances de notre architecture.



(a) (b) (c)

FIGURE 4.15 – Les images stéréo et la carte de disparité, a) et b) Une paire d'images stéréo avec différents plans de profondeur, c) Simulation de l'implémentation sur FPGA

Dans le dernier test nous utilisons des images d'extérieur acquises depuis un robot mobile (voir dans les figures 4.16a et 4.16b). L'image de disparité de la figure 4.16c montre des niveaux de gris clairs correspondant à une partie de la route plus proche de la scène. Les niveaux de gris plus sombres représentent des zones plus éloignées : nous pouvons noter que le camion blanc, sur les images originales, est représenté avec un niveau noir sur l'image de disparité, ce qui mérite une explication plus détaillée. Ceci

est dû au fait que les objets non texturés ne peuvent pas être facilement détectés par les algorithmes de vision stéréo. Ainsi, les pixels noirs représentent les pixels non identifiés et certains niveaux de gris apparaissant à l'intérieur du camion correspondent aux fenêtres latérales pour lesquelles il existe une texture. Une solution possible pour cette fausse détection serait d'utiliser une plus grande fenêtre de recherche dans la transformation census, mais comme nous l'avons déjà fait remarquer, cela a une incidence sur la complexité de l'architecture donc sur les performances globales. Malgré cela, c'est une possibilité qui pourrait être exploitée dans les applications de robotique mobile.

(a) (b) (c)

FIGURE 4.16 – Les images stéréo et la carte de disparité, a) et b) Une paire d'images d'extérieur acquises depuis un robot mobile, c) Carte de disparité résultant de l'implémentation sur FPGA

4.6 Conclusions

La technologie logique programmable se positionne donc comme une solution intermédiaire entre processeurs programmables et circuits VLSI dédiés, que ce soit au niveau du coût, de la performance et de la consommation électrique. Les frontières qui limitent son domaine d'utilisation sont donc beaucoup plus difficiles à cerner, celles-ci risquant de se modifier avec les évolutions technologiques à venir.

Actuellement, les FPGA permettent de développer des systèmes matériels qui répondent à la plupart des contraintes des applications temps réel. Cependant, le coût et le temps de conception des solutions fondées sur FPGA demeurent nettement plus élevés que les solutions logicielles. Les concepteurs doivent souvent répéter le flot de conception afin de répondre aux contraintes de performance de l'application tout en minimisant les ressources consommées par le circuit. Chaque itération de ce flot prend des heures ou des jours avant de converger vers une solution optimale.

La synthèse de haut niveau permet de réduire le temps de conception de l'algorithme avec une grande fiabilité du code généré ce qui fait que la qualité ne dépend pas de l'expérience et du talent du concepteur. Cependant, pour être efficace, la synthèse de haut niveau doit s'appuyer sur une méthode de conception qui tient compte de la spécificité des domaines d'application. Cela crée un lien que chaque outil de synthèse de haut niveau est développé pour un domaine particulier et les problèmes afférents, par conséquent cela limite le spectre d'action de chaque outil. Il est également nécessaire de rappeler que même dans la compilation sur logiciels, il y a encore une grande dépendance entre la manière dont un algorithme a été codé et la performance réalisée.

Dans l'état actuel, les outils de synthèse de haut niveau permettent de développer un système complet si les contraintes associées ne sont pas trop fortes, ce qui limite sérieusement leur utilisation dans beaucoup de domaines. Cependant, la synthèse de haut niveau reste un choix idéal pour l'exploration architecturale, laquelle peut fournir rapidement une architecture préliminaire qui peut servir de point de départ pour un processus de raffinement manuel et réduisant ainsi le temps de conception.

Chapitre 5

Architecture matérielle pour la détection d'obstacles

Sommaire

Ce chapitre est consacré au portage de l'algorithme pour la détection d'obstacles sur une plateforme matérielle afin de répondre aux contraintes temps réel qui nous sont imposées. Nous expliquerons dans un premier temps l'implémentation de chaque étape de l'algorithme de détection d'obstacles fondée sur les attributs de la couleur et de la texture et nous les évaluerons en termes d'architecture, de ressources consommées et de performances temps réel. Nous présenterons une analyse comparative entre nos architectures et celles issues de la littérature. Nous présenterons dans un deuxième temps l'extension de notre architecture à un système multi-caméras et la mise en œuvre de l'architecture globale pour la détection d'obstacles. Enfin, la mise en place du système de vision multi-caméras sur un robot de service sera explicitée.

5.1 Architecture pour la détection d'obstacles fondée sur l'apparence

L'architecture de détection d'obstacles fondée sur les attributs de couleur et de texture a été développée pour être implantée sur un circuit FPGA et est conçue comme un dispositif générique de traitement d'images à haute vitesse. Les critères d'optimisation retenus s'appuient sur une maximisation des performances en terme de traitement d'images par seconde et une minimisation des ressources consommées (mémoire, éléments logiques, multiplieurs embarqués) par le FPGA. Comme nous l'avons expliqué dans la section 3.2, notre algorithme de détection d'obstacles est composé de trois étapes principales : la transformation de couleur, le calcul des attributs de la texture et la classification des attributs de la couleur et de la texture.

La figure 5.1 montre l'architecture conçue pour mettre en œuvre la détection d'obstacles. Dans cette figure le premier module est le module de capture d'images, lequel assure l'interface entre la caméra et l'architecture pour la détection d'obstacles. En général ce module est fourni par le fabricant de la caméra, mais il est nécessaire de l'adapter au protocole de communication utilisé, à savoir le protocole Avalon. Ce module fournit un pixel au rythme de l'horloge caméra. Le module de capture fournit les données brutes depuis le capteur jusqu'au module de démosaïquage chargé de récupérer l'image couleur au rythme de l'horloge pixel. Les rythmes des horloges caméra et pixels peuvent êtres identiques mais dépendent de l'algorithme de démosaïquage utilisé pour récupérer l'image couleur. Le module de démosaïquage est indispensable et joue un rôle important même s'il ne fait pas implicitement partie de l'algorithme puisque c'est un processus nécessaire dans tous les algorithmes de traitement d'images fondés sur les attributs de couleur. Ce module fournit en sortie un bus de données qui transmet chaque pixel de l'image dans l'espace de couleur RGB. Chaque composante de couleur d'un pixel est codée sur n_b bits. La sortie du module de démosaïquage est liée aux trois modules principaux de l'architecture. Dans la figure 5.1, on peut voir en couleur verte le coeur de l'architecture représenté par ses trois modules principaux. Les architectures de ces trois modules et du module de démosaïquage seront détaillés dans les paragraphes suivants.

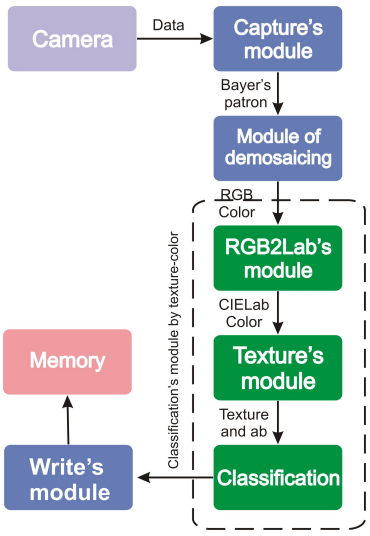


FIGURE 5.1 – Diagramme de l'architecture pour la détection d'obstacles fondée sur les attributs de la couleur et de la texture.

Dans la figure 5.1 le dernier module est le module d'écriture dans la mémoire. Ce module assure l'interface entre le protocole Avalon-ST et Avalon-MM. Il contient des mémoires internes pour assurer la continuité du flux d'entrée indépendamment des temps d'attente liés aux mémoires externes.

5.1.1 Architecture pour le démosaïquage

Le module de démosaïquage implémente l'algorithme d'interpolation qui construit l'image couleur dans l'espace RGB à partir de la mosaïque Bayer BGGR. Nous avons implémenté deux modules pour deux algorithmes d'interpolation différents : le démosaïquage par le plus proche voisin et le démosaïquage linéaire. Chaque module possède ses avantages et inconvénients selon les contraintes de l'application visée. Le principe de fonctionnement est similaire pour les deux modules : ceux-ci prennent en compte la position des pixels pour récupérer les trois composantes chromatiques. C'est à dire que les pixels bleus sont toujours positionnés dans les colonnes et les lignes impaires tandis que les pixels

rouges sont toujours positionnés dans les colonnes et les lignes paires. Enfin les pixels verts sont positionnés dans les colonnes impaires et les lignes paires ou les colonnes paires et les lignes impaires (voir figure 3.3).

Démosaïquage par le plus proche voisin

La figure 5.2 montre l'architecture du module de démosaïquage par le plus proche voisin. Cette architecture utilise une fenêtre glissante de 2×2 pixels pour récupérer les trois composantes chromatiques. La fenêtre glissante est construite en utilisant deux registres à décalage de profondeur de n_b bits et une mémoire de taille de $r_{uc} \times n_b$ bits, où r_{uc} représente le nombre de colonnes ou la résolution dans la coordonnée u de l'image délivrée par la caméra.

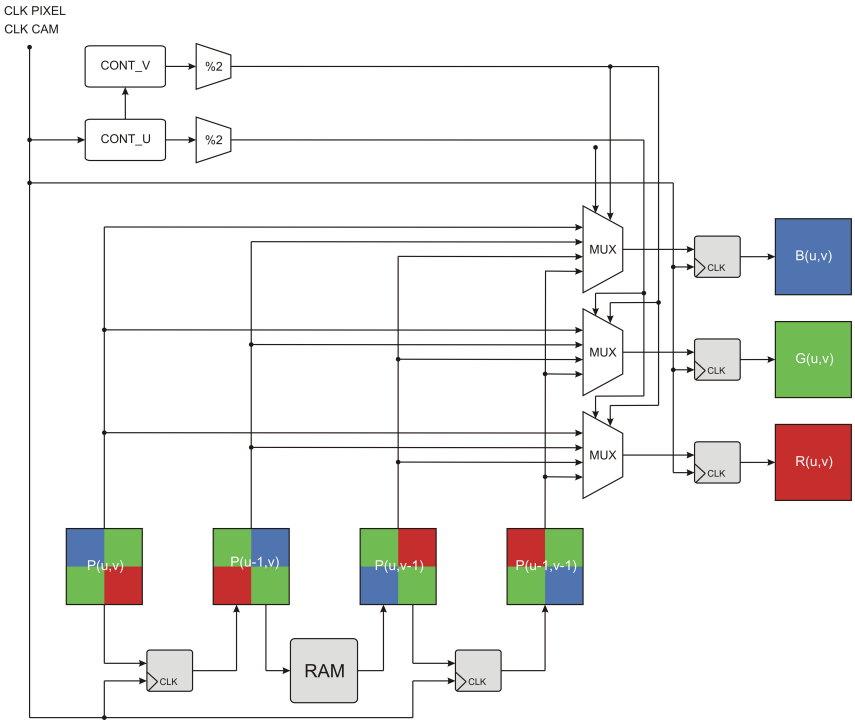


FIGURE 5.2 – L'architecture du module de calcul de démosaïquage par le plus proche voisin.

Le pixel courant $P(u, v)$, en bas à gauche dans la figure 5.2 est affecté à une des quatre couleurs possibles de la mosaïque Bayer BGGR (bleu, vert, vert ou rouge) selon sa position (u, v). A chaque top de l'horloge caméra la valeur d'intensité est stockée dans les registres et la mémoire de sorte que le premier registre mémorise le pixel $P(u-1, v)$ de couleur verte, bleue, rouge ou verte. Le port d'entrée de la mémoire permet d'avoir le pixel $P(u-2, v)$ de couleur bleue, verte, verte ou rouge et le port de sortie le pixel $P(u, v-1)$ de couleur verte, rouge, bleue ou verte. Cette mémoire fonctionne comme une FIFO qui est toujours presque pleine, ce qui explique un décalage d'une ligne moins un pixel entre le port d'entrée et le port de sortie. Le second registre permet de mémoriser le pixel $P(u-1, v-1)$ de couleur rouge, verte, verte ou bleue. Les trois composantes chromatiques rouge $R(u, v)$, verte $G(u, v)$ et bleue $B(u, v)$ du pixel sont sélectionnées depuis la fenêtre glissante en fonction de sa position (u, v) et à l'aide de deux compteurs et trois blocs de multiplexage. Le premier compteur $CONT\_U$ indique le numéro de colonne tandis que le deuxième compteur $CONT\_V$ indique le numéro de ligne. La parité des deux compteurs est évaluée pour savoir si la colonne et la ligne sont paire ou impaire. Les résultats des comparaisons de parité sont utilisés comme signaux de commande des multiplexeurs de sorte que le

premier bloc, de haut en bas dans la figure 5.2, donne toujours la composante chromatique bleue $B(u,v)$, le deuxième la composante chromatique verte $G(u,v)$ et le troisième la composante chromatique rouge $R(u,v)$.

Ce module de démosaïquage fournit une image couleur de qualité mediocre mais cependant il ne réduit pas la taille de l'image et l'horloge caméra est la même que l'horloge pixel : le débit vidéo se trouve maintenu.

Démosaïquage linéaire

La figure 5.3 représente l'architecture du module de démosaïquage linéaire. Cette architecture est similaire à celle du module de démosaïquage par le plus proche voisin. Toutefois, dans cette architecture quatre blocs de multiplexage au lieu de trois sont utilisés. Le premier bloc de multiplexage, de haut en bas dans la figure 5.3, donne la composante chromatique bleue $B(u,v)$. La composante chromatique verte $G(u,v)$ est obtenue par la somme des sorties des blocs de multiplexage deux et trois. Le quatrième bloc de multiplexage donne la composante chromatique rouge $R(u,v)$. Il faut souligner que la fréquence de l'horloge caméra est quatre fois plus rapide que la fréquence de l'horloge pixel, ce qui ralentit le débit vidéo mais améliore la qualité de l'image obtenue par rapport au démosaïquage par le plus proche voisin.

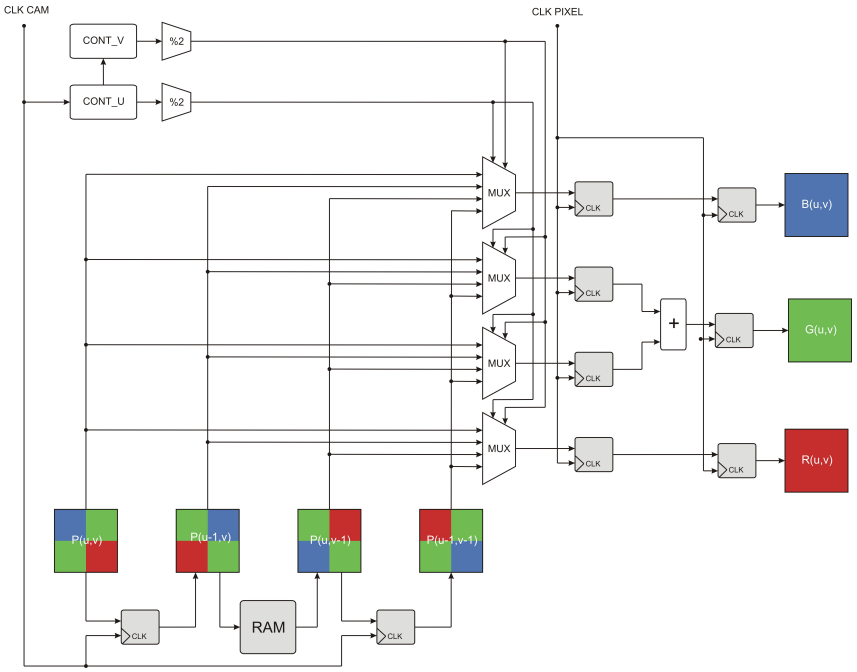


FIGURE 5.3 – L'architecture du module de calcul de démosaïquage linéaire.

5.1.2 Architecture pour la transformation d'espace de couleur

La transformation de l'espace de couleur est faite en trois étapes : la première est la transformation linéaire de l'espace de couleur *RGB* dans l'espace de couleur de coordonnées primaires *XYZ*, cette transformation est obtenue par l'equation 3.9. Ensuite l'espace de couleur de coordonnées primaires *XYZ* est normalisé par le blanc de référence pour obtenir l'espace de couleur normalisé *XYZ*. Enfin l'espace de couleur *CIE-L*ab* est obtenue par la transformation non-linéaire 3.13 de l'espace de couleur normalisé *XYZ*. Afin de réduire le temps de latence et de simplifier l'architecture du module de transformation

de couleur, nous avons fusionné les deux premières étapes de sorte que l'espace de couleur *RGB* est directement transformé dans l'espace de couleur normalisé *XYZ* en appliquant l'équation suivante :

$$\begin{pmatrix} X/X_w \\ Y/Y_w \\ Z/Z_w \end{pmatrix} = \begin{bmatrix} 0.433953 & 0.376219 & 0.189828 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.017758 & 0.109476 & 0.872766 \end{bmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{5.1}$$

Cette matrice de transformation est obtenue par la division de la matrice de transformation 3.9 avec le blanc de référence puis codée sur n_b bits. Dans notre cas elle est codée sur 8 bits, ce qui correspond au nombre de bits pour chaque composante chromatique de l'espace de couleur RGB délivrée par le module de démosaïquage. La matrice codée est la suivante :

$$\begin{bmatrix} CRX & CGX & CBX \\ CRY & CGY & CBY \\ CRZ & CGZ & CBZ \end{bmatrix} = \begin{bmatrix} 111 & 96 & 48 \\ 54 & 182 & 19 \\ 4 & 28 & 223 \end{bmatrix} \tag{5.2}$$

La figure 5.4 montre l'architecture mise en œuvre pour obtenir la transformation de couleur. Les trois premières étapes dans le pipeline calculent la transformation de l'espace couleur *RGB* dans l'espace de couleur normalisé *XYZ*. Les constantes $CRX, CGX, CBX, CRY, CGY, CBY, CRZ, CGZ$ et CBZ sont les neuf éléments de la matrice de transformation 5.2. Les sorties des trois multiplieurs sont codées sur $2 \cdot n_b$ bits et la matrice de transformation donnant un espace de couleur normalisé, les sommes des sorties des multiplieurs sont toujours codés avec le même nombre de bits : $2 \cdot n_b$. De même les entrées et les sorties des six additionneurs dans les deuxième et troisième étapes du pipeline sont codées sur $2 \cdot n_b$.

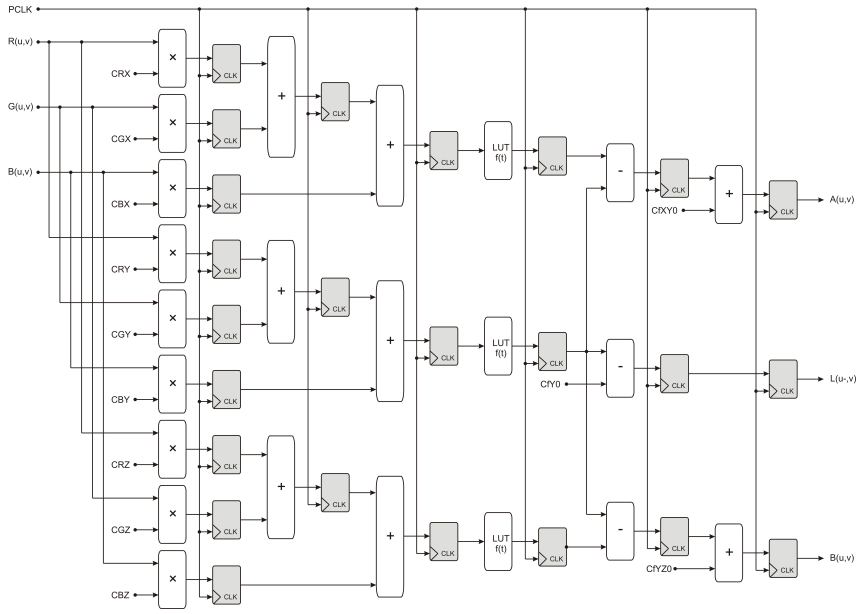
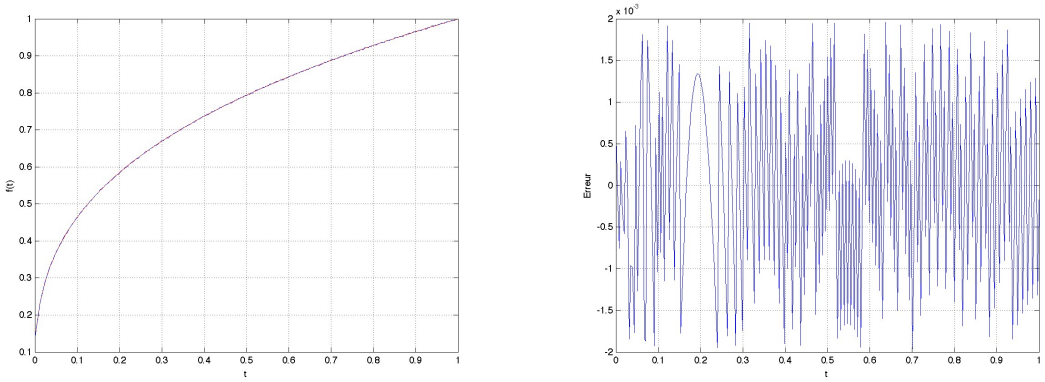


FIGURE 5.4 – L'architecture du module pour calculer la transformation CIEL*ab.

Dans la figure 5.4, les trois dernières étapes du pipeline calculent la transformation de l'espace de couleur normalisé *XYZ* dans l'espace *CIE-L*ab*, la transformation non linéaire de l'espace *XYZ* étant stockée dans trois LUTs. La figure 5.5a montre la fonction originale et sa fonction correspondante codée sur 8 bits. L'erreur de la fonction codée sur 8 bits est présentée dans la figure 5.5b. Cette fonction a une erreur quadratique moyenne de 0.001134, une erreur quadratique normalisée de 0.001316 et un coefficient de variation de l'erreur quadratique moyenne de 0.001513 , négligeables pour la plupart des

applications, et dont l'impact peut être atténué voire supprimé par le module de classification. Dans la dernière étape, les constantes $CfY0$, $CfXY0$ et $CfYZ0$ sont utilisées pour normaliser l'espace de couleur *CIE-L*ab*. Ainsi, la sortie $L*$ de ce module, qui représente la luminance, est utilisée pour calculer la texture, et les composantes chromatiques a et b sont utilisées comme attributs de couleurs dans la classification. Le temps de latence de ce module est de six tops d'horloge pixel.



(a) (b)

FIGURE 5.5 – (a) Le graphe de la transformation non-linéaire $f\,(t)$, en couleur rouge la fonction originale et en couleur bleue la fonction codée sur 8 bits, (b) Le graphe de l'erreur entre la fonction originale et la fonction codée sur 8 bits.

5.1.3 Architecture pour l'analyse de texture

L'architecture pour l'analyse de texture dense est composée de deux processus principaux : la somme et la différence des images et le calcul des attributs de texture. Ces processus sont représentés à l'intérieur de deux rectangles pointillés dans la figure 5.6. Le rectangle pointillé en haut représente le module de la somme et la différence des images, celui du bas représente le module de calcul des attributs de texture divisé lui-même en six modules. Cinq de ces modules (en couleur verte dans la figure 5.6) représentent les cinq premiers attributs de texture dans la table 3.2. Le sixième module, appelé attribut de texture de pseudo-variance, est un module intermédiaire nécessaire pour calculer les attributs de texture de la variance et la corrélation. La mise en œuvre sur matériel de ces modules est détaillée dans les paragraphes suivants.

Module de sommes et différences d'images

La somme et la différence des images est le premier processus exécuté lors de l'analyse de texture dense. L'addition et la soustraction sont des opérations effectuées entre le pixel courant et les pixels déplacés. Seize déplacements sont utilisés au total pour quatre distances et quatre orientations, telles que (1, 2, 3 et 4 pixels) et ($45°$, $90°$, $135°$ et $180°$) respectivement. L'orientation de $0°$ est remplacé par $180°$ afin d'éviter un délai supplémentaire d'attente causé par le pixel situé à droite, c'est à dire à $0°$. Un schéma du module de l'architecture proposée est représenté dans la figure 5.7. Les registres à décalage de profondeur de n_b bits et les mémoires RAM sont utilisés pour retarder le pixel courant et obtenir les déplacements. La première mémoire a une taille de $(r_u - 4) \times n_b$ bits, où r_u représente la résolution dans la coordonnée u de l'image délivrée par le module de démosaïquage. La deuxième mémoire a une taille de $(r_u - 2) \times n_b$ bits, la troisième une taille de $(r_u - 4) \times n_b$ bits et la quatrième une taille de $(r_u - 6) \times n_b$ bits. Le temps de latence de ce module est de un top d'horloge pixel. Pour

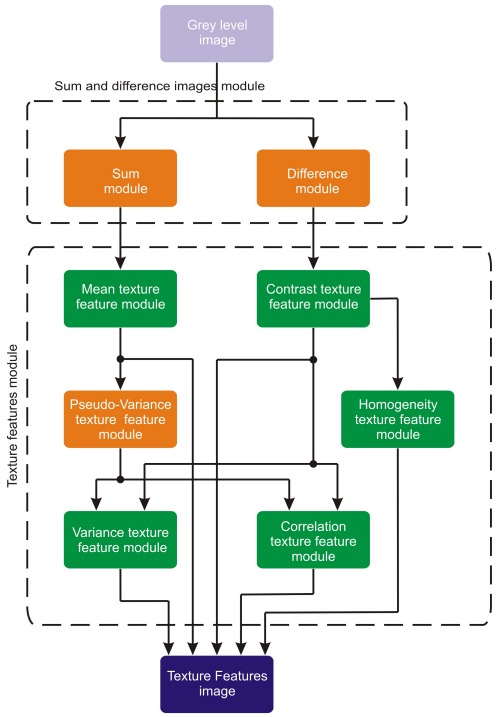


FIGURE 5.6 – Diagramme de l'architecture pour l'analyse de texture fondée sur l'adéquation des histogrammes de sommes et différences.

chaque déplacement, une fenêtre de traitement de $K \times L = 17 \times 15$ pixels est utilisée pour calculer les attributs de texture de la moyenne, la variance, l'homogénéité, le contraste et la corrélation. La taille du pixel de sortie de ce module est de $n_b + 2$, 1 bit pour le signe et l'autre pour l'opération d'addition ou de soustraction.

Module de l'attribut de texture de la moyenne

L'attribut de la moyenne (voir équation 5.3) est l'addition de tous les pixels sur l'image de la luminance L_S dans une fenêtre de traitement de taille $K \times L = 17 \times 15$ pixels, où K et L représentent le nombre de colonnes et de lignes, respectivement, dans la fenêtre de traitement.

$$\mu = \frac{1}{2N} \sum_{k \in D} \sum_{l \in D} L_S(k, l) \tag{5.3}$$

Pour réaliser le calcul de l'attribut de moyenne en temps réel à partir d'un flot de pixels issu de l'étape de la somme de l'image, trois méthodes sont possibles.

La première méthode calcule l'attribut de moyenne à partir d'une phase d'addition, de stockage intermédiaire et d'une phase de décalage comme représenté dans la figure 5.8a à titre d'exemple pour une fenêtre de $K \times L = 3 \times 3$ pixels. Cette méthode nécessite $(K-1) \times L$ registres à décalage de profondeur de $n_b + 2$ bits, $L - 1$ mémoires de stockage intermédiaire et $(K \times L) - 1$ additionneurs parallèles disposés en pyramide. La taille de chaque mémoire est de $(n_b + 2) \times (r_u - K + 1)$ bits. Le bloc de mémoires et de registres à décalage sont câblés de telle sorte qu'ils aient en mémoire les pixels de la fenêtre de calcul courante. A chaque top de l'horloge pixel une nouvelle fenêtre est chargée dans le bloc et les additionneurs fournissent un résultat image de la valeur moyenne de la fenêtre. Cette méthode a un temps de latence de $\lceil \log_2 (K \times L) \rceil$ tops de l'horloge pixel.

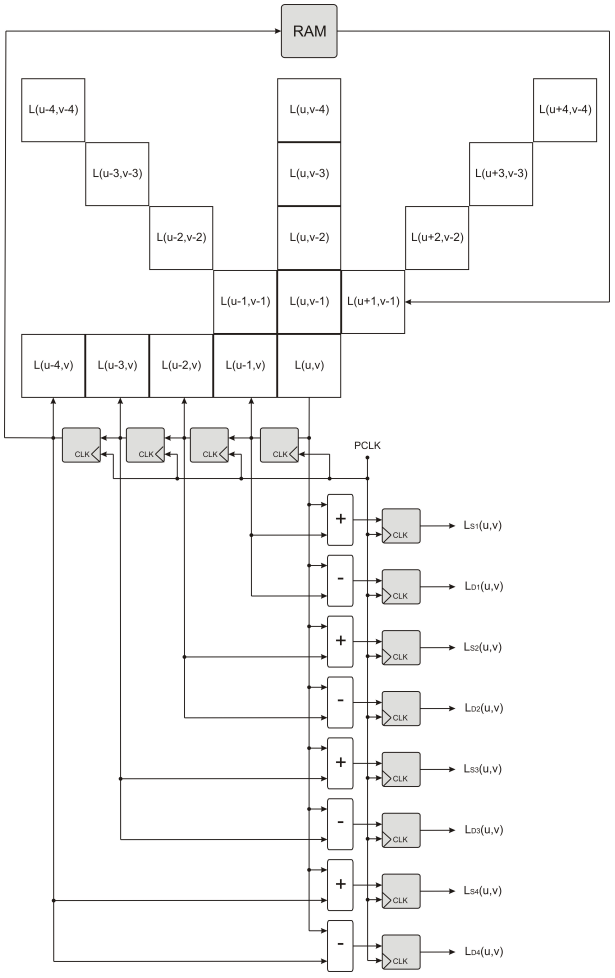
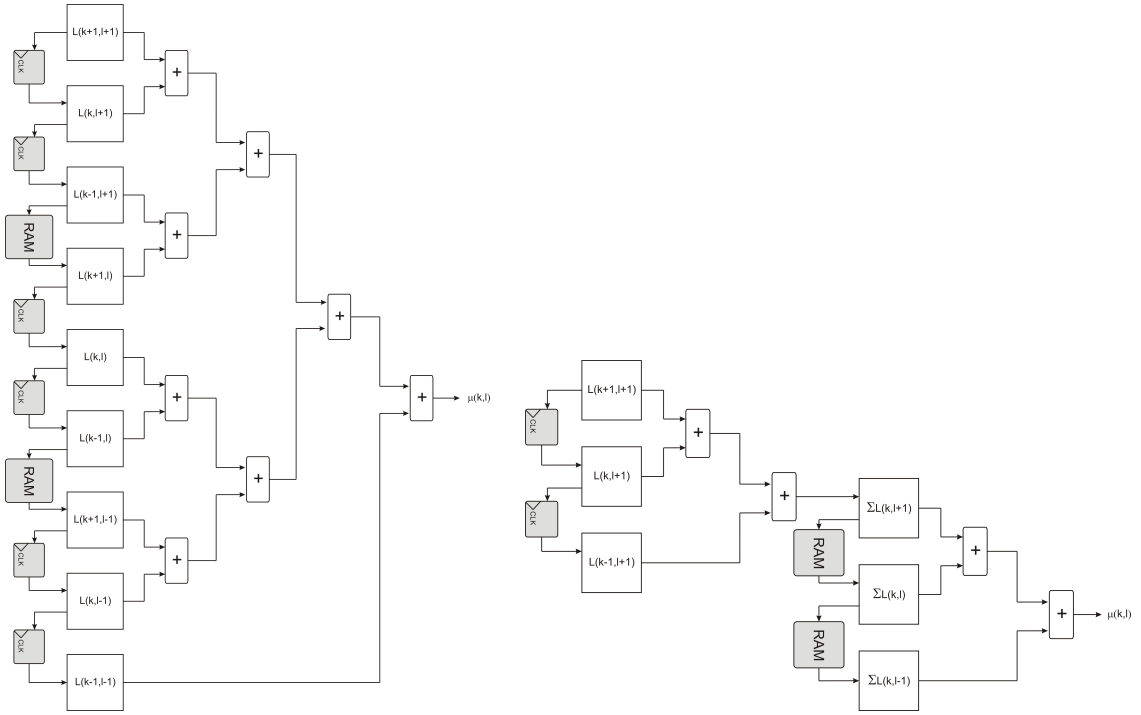


FIGURE 5.7 – L'architecture du module pour calculer la somme et la différence des images.

La deuxième méthode effectue le calcul de l'attribut de moyenne en deux étapes d'addition et une étape de stockage intermédiaire : ainsi on calcule la somme horizontale de K pixels successifs puis la somme verticale à partir des pixels adjacents comme représenté dans la figure 5.8b à titre d'exemple pour une fenêtre de $K \times L = 3 \times 3$ pixels. Cette méthode nécessite donc $K-1$ registres à décalage de profondeur de $n_b + 2$ bits, $L-1$ mémoires de stockage intermédiaire et $K+L-2$ additionneurs parallèles. La taille de chaque mémoire est de $(n_b + k + 2) \times r_u$ bits. A chaque top de l'horloge pixel la chaîne de $K-1$ registres à décalage mémorise les $K-1$ pixels successifs appartenant à la même ligne. Les $K-1$ valeurs successives et le pixel courant sont simultanément additionnées pour fournir une image de l'attribut moyenne de ces K pixels. Ce résultat est stocké ensuite dans une mémoire intermédiaire qui fait un décalage d'une ligne entre le port d'entrée et le porte de sortie. Les $L-1$ sorties des mémoires et l'addition horizontale courante sont simultanément additionnées pour former la somme de $K \times L$ pixels correspondant à la fenêtre de traitement courante. Cette méthode a un temps de latence de $\max\left(\lceil \log_2(K) \rceil, \lceil \log_2(L-1) \rceil\right) + 1$ tops de l'horloge pixel.

La troisième méthode calcule l'attribut de moyenne à partir de deux étapes d'addition et une étape de stockage intermédiaire : ainsi on calcule la valeur de l'attribut de moyenne courante en réutilisant la valeur de l'attribut de moyenne précédent comme représenté dans la figure 5.9. Cette méthode nécessite $K-1$ registres à décalage de profondeur de $n_b + 2$ bits, $L-1$ mémoires de stockage intermédiaire



(a) (b)

FIGURE 5.8 – Diagrammes des architectures pour calculer l'attribut de moyenne (a) Première méthode, (b) Deuxième méthode.

et $2L$ additionneurs parallèles disposés en pyramide. La taille de chaque mémoire double-port est de $(n_b + 2) \times r_u$ bits. Le premier port est à lecture et écriture alors que le second est à lecture seulement. Les deux ports de lecture de chaque mémoire ont un décalage de K pixels entre eux, tandis que le premier port de lecture de la première mémoire est câblé de telle sorte qu'avec le premier port d'écriture de la deuxième mémoire il y ait un décalage d'une ligne ou r_u pixels entre eux. La même structure est répétée de telle sorte que le premier port de lecture de la deuxième mémoire est câblé au premier port d'écriture de la troisième mémoire, et ainsi de suite pour les autres mémoires. Cette structure génère une sorte de fenêtre glissante de taille de $K + 1 \times L$ dans laquelle la dernière colonne de la fenêtre est dans le premier port de lecture des mémoires et la première colonne est dans le second port de lecture (en couleurs bleue et rouge dans la figure 5.9, respectivement). Dans la figure 5.9, la valeur courante de l'attribut de moyenne dans la fenêtre de traitement est l'addition de la région en couleur grise et la dernière colonne de la fenêtre glissante, alors que la valeur précédente de l'attribut de moyenne dans la fenêtre de traitement est l'addition de la région grise et la première colonne de la fenêtre glissante. Cette région grise représente les colonnes communes dans la fenêtre glissante pour la valeur courante et la valeur précédente de l'attribut. Ainsi, la valeur courante de l'attribut de moyenne dans la fenêtre de traitement est calculée en additionnant la somme des valeurs dans la dernière colonne de la fenêtre glissante à la valeur précédente de l'attribut puis en soustrayant la somme des valeurs dans la première colonne de la fenêtre glissante au résultat de l'opération précédente.

Afin d'optimiser le temps de latence dans la figure 5.9, on a fait l'addition des valeurs dans la dernière colonne de la fenêtre glissante et la soustraction des valeurs dans la première colonne de la fenêtre glissante en utilisant une architecture pyramidale. Les résultats de l'addition et la soustraction sont additionnés entre eux, ce résultat est à son tour sommé avec la valeur précédente de l'attribut de moyenne. Cette valeur précédente de l'attribut de moyenne est stockée dans le dernier registre à décalage qui est

câblé en rétroaction pour obtenir la valeur courante de l'attribut. Cette méthode a un temps de latence de $\lceil \log_2 (L) \rceil + 2$ tops de l'horloge pixel.

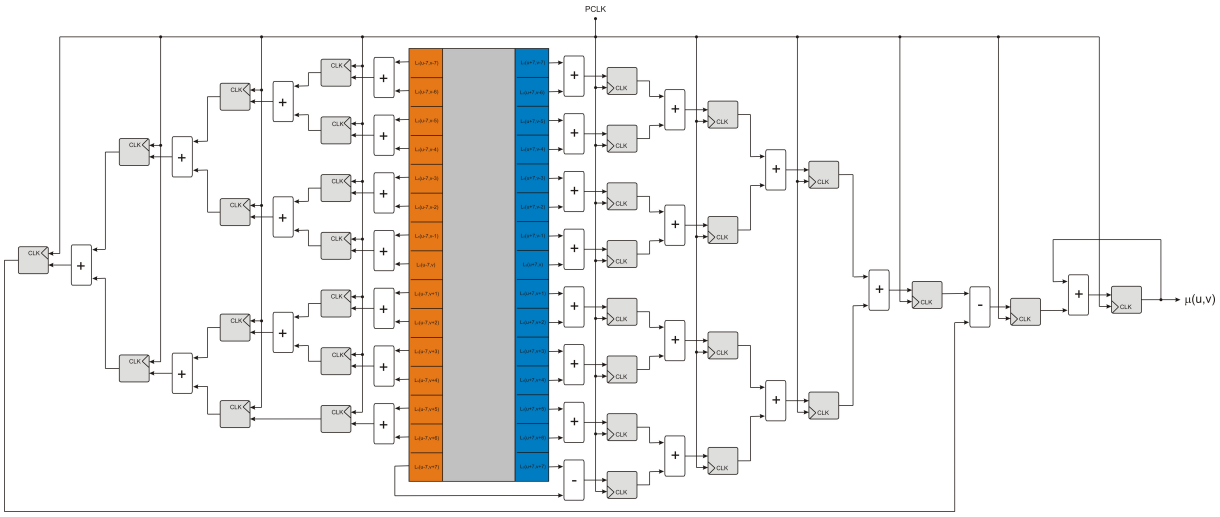


FIGURE 5.9 – L'architecture du module pour calculer l'attribut de texture de la moyenne.

Nous avons retenu comme architecture la troisième méthode car elle consomme d'une part moins de ressources et présente d'autre part un temps de latence inférieur. C'est en raison de la taille de la fenêtre de traitement que la troisième méthode nécessite moins de ressources de stockage que la deuxième méthode.

Module de l'attribut de texture du contraste

L'attribut du contraste (voir équation 5.4) est similaire à l'attribut de moyenne sauf que l'image de somme de la luminance L_S est remplacée par l'image de différence de la luminance L_D. Alors, l'attribut de contraste est l'addition de tous les pixels carrés sur l'image de différence de la luminance L_S à l'intérieur de la fenêtre de traitement de taille $K \times L = 17 \times 15$ pixels.

$$\frac{1}{N} \sum_{k \in D} \sum_{l \in D} L_D \left(k, l\right)^2 \tag{5.4}$$

Une partie de l'architecture du module du contraste est représentée dans le rectangle supérieur pointillé de la figure 5.10. Pour réaliser ce calcul en temps réel à partir d'un flot de pixels issu de l'étape de la différence de l'image, cette architecture utilise une méthode similaire à l'architecture du module de l'attribut de moyenne. Toutefois, les différences essentielles dans leurs architectures sont dues aux multiplieurs supplémentaires placés entre l'architecture pyramidale d'addition et la fenêtre glissante de l'image de différence de la luminance L_D. Ainsi, les multiplieurs de $n_b + 2$ bits sont ajoutés aux ressources nécessaires et le temps de latence associé est de $\lceil \log_2 (L) \rceil + 3$ tops de l'horloge pixel.

Module de l'attribut de texture de l'homogénéité

L'attribut de l'homogénéité (voir équation 5.5) réutilise la valeur du pixel carré de l'attribut du contraste. Cet attribut est l'addition de l'inverse de tous les pixels carrés sur l'image de différence de la luminance L_S à l'intérieur de la fenêtre de traitement de taille $K \times L = 17 \times 15$ pixels. Le pixel carré

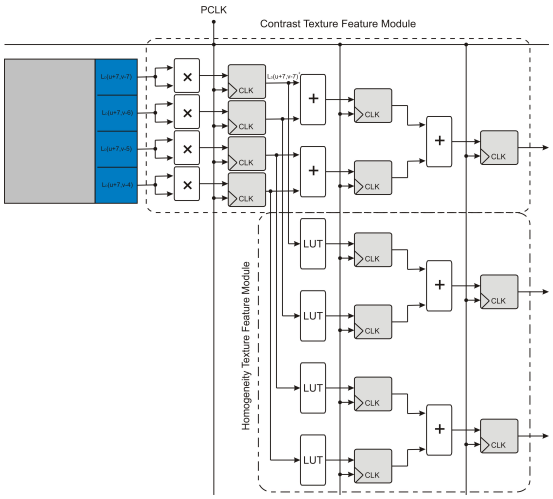


FIGURE 5.10 – L'architecture des modules pour calculer les attributs de texture du contraste et l'homogénéité.

de l'image de différence de la luminance L_S est incrémenté de un pour éviter une éventuelle indétermination liée à une division par zéro dans l'équation 5.5. Cette équation donnant l'attribut de l'homogénéité normalisé à 1, nous avons changé l'échelle pour le calculer normalisé à 2^{n_b-1}.

$$\frac{1}{N}\sum_{k\in D}\sum_{l\in D}\frac{1}{L_D\left(k,l\right)^2+1} \tag{5.5}$$

Le rectangle inférieur en pointillé dans la figure 5.10 présente une partie de l'architecture du module d'attribut de l'homogénéité. Ce module réutilise les résultats des multiplieurs du module du contraste, c'est à dire les valeurs des pixels carrés sur l'image de différence de la luminance L_D^2. L'inverse de ces valeurs est ensuite obtenu à partir des LUTs, lesquelles sont adressées par les valeurs des pixels élevées au carré. Enfin, les sorties des LUTs sont câblées à l'architecture pyramidale de sommation pour obtenir l'attribut d'homogénéité. Ce module ajoute les $2 \times L$ LUTs aux ressources nécessaires et a un temps de latence de $\lceil \log_2 (L) \rceil + 4$ tops de l'horloge pixel.

Module de l'attribut de texture de la pseudo-variance

L'attribut de pseudo-variance est une valeur intermédiaire requise pour calculer les attributs de la variance et la corrélation. L'attribut de pseudo-variance est défini par l'équation 5.6. Cet attribut, contrairement aux attributs décrits ci-dessus, ne peut être calculé par la troisième ou la deuxième méthode expliquées dans la section de l'architecture du module de la moyenne, car il n'y a pas de région commune entre les fenêtres de traitement courante et précédente. En d'autres termes, il n'y a pas de chevauchement entre les valeurs des fenêtres de traitement. Cela vient du fait que l'attribut de la moyenne dans la fenêtre de traitement courante doit être soustrait à chaque pixel de cette fenêtre et la valeur de l'attribut de la moyenne changeant pour chaque fenêtre, sa réutilisation devient impossible. Il en résulte que seule la première méthode permet le calcul de cet attibut.

$$\frac{1}{2N}\sum_{k\in D}\sum_{l\in D}\left(L_S\left(k,l\right)-2\mu\left(k,l\right)\right)^2 \tag{5.6}$$

La figure 5.11 illustre une partie de l'architecture du module de calcul de l'attribut de la pseudo-variance. Ce module utilise la dernière colonne de la fenêtre glissante et la dernière colonne de la fenêtre

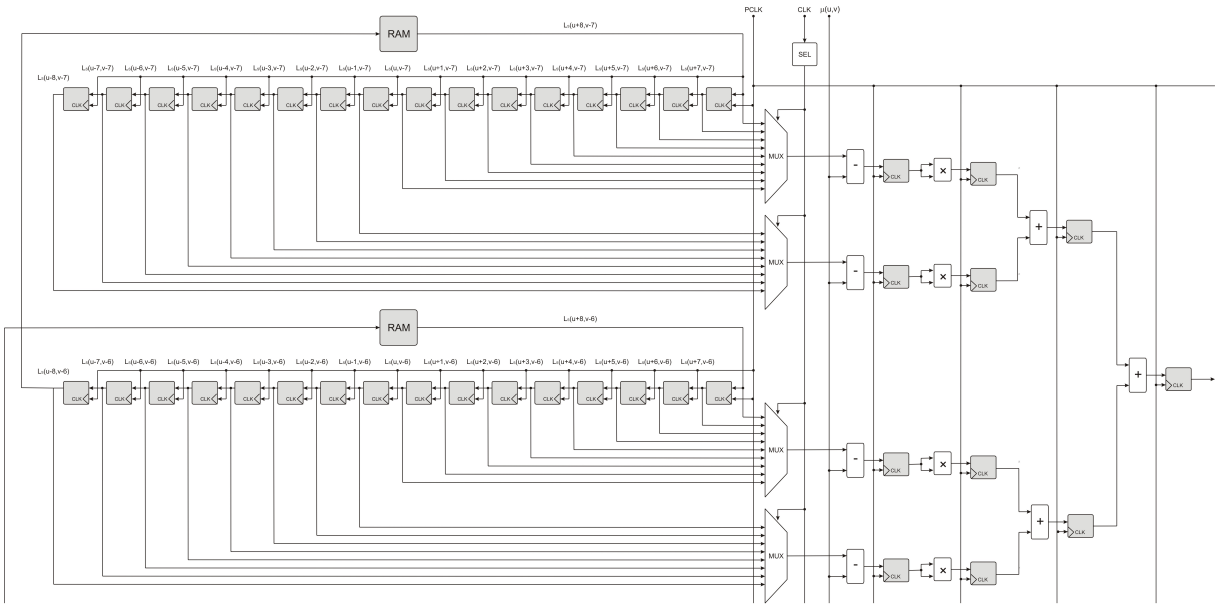


FIGURE 5.11 – L'architecture du module pour calculer l'attribut de texture de la pseudo-variance.

de traitement courant pour construire sa propre fenêtre de traitement en utilisant $K \times L$ registres à décalage de profondeur $n_b + 2$ bits. Les registres à décalage sont câblés de telle sorte qu'ils aient en mémoire les pixels de la fenêtre de traitement courante. A chaque top de l'horloge pixel une nouvelle fenêtre de traitement est chargée dans les registres. Les mémoires RAMs de décalage entre lignes sont partagées avec les autres modules afin de réduire les ressources. Afin de les optimiser davantage, ce module réutilise les multiplieurs embarqués au détriment de la performance qui se trouve alors sensiblement réduite. La contrainte sur la performance est de 30 fps mínimum avec une taille d'image de 640×480 pixels, de sorte que la fréquence pixel minimum doit être de $9,216$ kHz (sans temps d'attente). Dans notre cas, nous avons fixé la fréquence pixel à $f_{PCLK} = 10$ MHz pour simplifier l'architecture et parallèlement cette fréquence permet d'atteindre les objectifs fixés, y compris les temps d'attente. Pour préserver la portabilité des modules, nous avons utilisé une fréquence maximale de fonctionnement des multiplieurs embarques de $f = 100$ MHz : il est donc possible de réutiliser au maximum 10 fois chaque multiplieur par top d'horloge pixel. Ensuite, la fenêtre de traitement étant de $K \times L = 17 \times 15$ pixels et pouvant être balayée par les colonnes ou les lignes, nous avons minimisé le nombre de multiplieurs embarqués tout en maximisant leur utilisation. Pour la première hypothèse, nous appliquons l'équation 5.7, qui sans notre cas n'est pas discriminatoire puisque les deux colonnes et les lignes utilisent le même nombre de multiplieurs embarqués, $\#_\times = 2$ dans notre cas. Ensuite nous appliquons la seconde hypothèse (voir l'équation 5.8), ce qui fait que dans notre cas nous balayons les 17 colonnes de la fenêtre de traitement, dont au moins 2 pixels doivent être traités simultanément pour chaque ligne (voir la figure 5.11). Dans la figure 5.11, les multiplexeurs et le compteur font le balayage des colonnes pour récupérer les pixels. Ces pixels sont alors soustraits à l'attribut de la moyenne sur la fenêtre de traitement courante, puis le résultat est élevé au carré, pour finalement alimenter une architecture pyramidale de sommation et obtenir ainsi l'attribut de la pseudo-variance. Une machine à états permet de passer les résultats entre lignes pour obtenir la valeur de l'attribut. Ce module a un temps de latence d'un top de l'horloge pixel, toutefois ayant besoin de la valeur de l'attribut de moyenne, il se trouve ramené à $\lceil \log_2 (L) \rceil + 3$ tops de l'horloge pixel.

$$\#_\times = \min\left(\left\lceil \frac{K \times f_{PCLK}}{f} \right\rceil, \left\lceil \frac{L \times f_{PCLK}}{f} \right\rceil\right) \tag{5.7}$$

$$\#_{\#} = \max\left(\left\lceil \frac{K}{\#_{\times}} \right\rceil, \left\lceil \frac{L}{\#_{\times}} \right\rceil\right) \tag{5.8}$$

Modules des attributs de la variance et de la corrélation

Les attributs de la variance et de la corrélation sont définis par les équations 5.9 et 5.10. Ces deux attributs peuvent être redéfinis comme étant la somme et la différence des attributs de la pseudo-variance et le contraste, respectivement. Cela simplifie la complexité de calcul de ces attributs.

$$\frac{1}{2N}\left(\sum_{k\in D}\sum_{l\in D}\left(L_S\left(k,l\right)-2\mu\left(k,l\right)\right)^2+\sum_{k\in D}\sum_{l\in D}L_D\left(k,l\right)^2\right) \tag{5.9}$$

$$\frac{1}{2N}\left(\sum_{k\in D}\sum_{l\in D}\left(L_S\left(k,l\right)-2\mu\left(k,l\right)\right)^2-\sum_{k\in D}\sum_{L\in D}L_D\left(k,l\right)^2\right) \tag{5.10}$$

La figure 5.12 présente les architectures des modules de calcul des attributs de la variance et de la corrélation. Ces architectures se réduisent à deux opérations : une addition et une soustraction une fois que les attributs de la pseudo-variance et le contraste ont été calculés. Le temps de latence de ces deux modules est d'un top d'horloge pixel plus celui lié à la pseudo-variance, ce qui donne un temps de latence de $\lceil \log_2(L) \rceil + 4$ tops de l'horloge pixel.

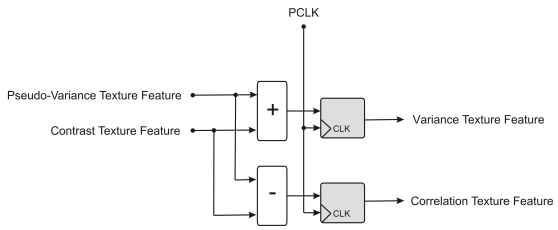


FIGURE 5.12 – L'architecture des modules pour calculer les attributs de la variance et de la corrélation.

Afin de synchroniser les architectures des modules des attributs de texture, nous prenons en compte le temps de latence le plus long, ce qui correspond aux modules des attributs de l'homogénéité, la variance et la corrélation dont la valeur est de $\lceil \log_2(L) \rceil + 4$ tops de l'horloge pixel. Les valeurs de sortie des modules des attributs de la moyenne, du contraste et de la pseudo-variance sont retardés en utilisant des registres à décalage pour synchroniser le module global d'analyse de la texture. Le temps de latence de ce module global est de $\lceil \log_2(L) \rceil + 4$ tops de l'horloge pixel.

5.1.4 Architecture pour l'algorithme de classification

L'architecture du module de classification dense consiste en un classificateur fondé sur l'algorithme AdaBoost. Ce classificateur est obtenu à partir de l'algorithme d'apprentissage hors ligne décrit dans la sous-section 3.2.5. Dans cette sous-section, nous présentons tout d'abord l'architecture du module de l'hypothèse faible suivi de l'architecture du module du classificateur basé sur l'algorithme AdaBoost. Cette dernière architecture utilise le module de l'hypothèse faible. Enfin, nous décrirons le processus mis en place pour automatiser l'apprentissage et la génération du module de classification.

Module de l'hypothèse faible

La technique des arbres de décision est utilisée comme base pour le module de l'hypothèse faible. Une profondeur de $n_t = 5$ est utilisée dans chaque arbre de decision : on a fixé cette valeur dans tous les arbres de décision afin de réduire la complexité des modules de plus haut niveau dans l'architecture de classification et pour définir un seul module reconfigurable pour tous les arbres de décision. La figure 5.13 montre une partie de l'architecture du module de l'arbre de décision. Ce module a comme entrée le bus de données x de taille $n_f \times n_b$ bits, où n_f représente le nombre des attributs codés sur n_b bits. Le nombre des attributs est $6 \times n_d + 2$, où n_d représente le nombre de déplacements dans les attributs de texture, 6 étant le nombre d'attributs de texture par déplacement et 2, le nombre d'attributs de la couleur obtenus à partir de la transformation CIE-Lab. L'architecture matérielle de ce module se compose de 1 niveau de sélection et 6 niveaux de comparaisons. Le niveau de sélection consiste à $2^{n_t+1} - 1 = 63$ multiplexeurs qui choisissent les attributs sur le bus x pour les comparer dans chaque branche particulière de l'arbre de décision. Les multiplexeurs sont commandés par le bus J, lequel est contrôlé par le module de plus haut niveau. Chaque niveau de comparaison, sauf le premier, est composé d'un étage de multiplexage et d'un étage de comparaison. L'étage de multiplexage est utilisé pour choisir l'attribut et le seuil qui doivent être comparés. Les multiplexeurs sont commandés par les résultats des niveaux précédents de comparaison, puisque cela détermine la branche de l'arbre où se situe le pixel courant défini par ses attributs sur le bus x. Les premiers 5 niveaux de comparaison correspondent aux n_t niveaux de profondeur de l'arbre de décision, le dernier niveau de comparaison est la sortie de l'arbre de décision qui est différente pour chaque feuille de l'arbre. Les seuils C sont commandés par le bus du même nom, ce bus est aussi contrôlé par le module de plus haut niveau. Le temps de latence de ce module est de $2 \times (n_t + 1) = 12$ tops de l'horloge pixel.

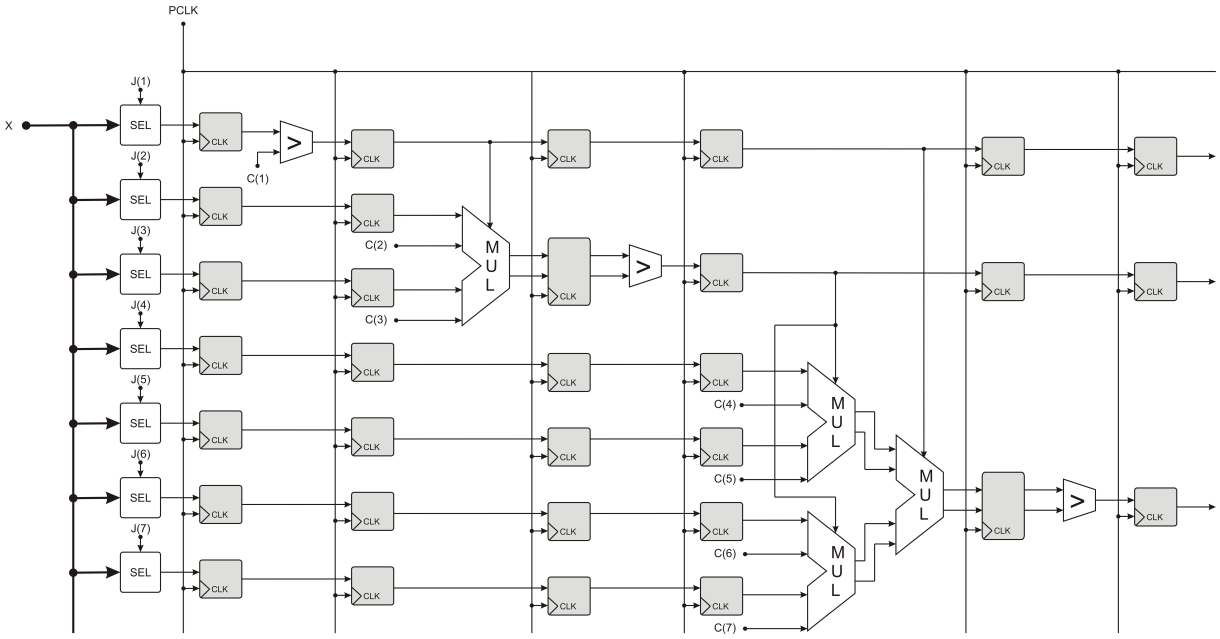


FIGURE 5.13 – L'architecture du module du classificateur faible implémentant un arbre de décision.

Module de l'algorithme Adaptive Boosting

La figure 5.14 montre une partie de l'architecture du module de classification basé sur l'algorithme Adaptive Boosting qui est constitué de T modules d'hypothèses faibles. Le module AdaBoost configure

les bus : J pour les attributs dans les sélecteurs et C pour les seuils dans les multiplexeurs. Pour chacun des modules d'hypothèses faibles, cela revient à configurer $2^{n_t+2} - 2 = 126$ valeurs automatiquement pour réduire les erreurs possibles. La décision finale de chaque hypothèse faible est multipliée par le poids α correspondant, ce poids étant généré et câblé automatiquement à partir de la méthode d'apprentissage mise en oeuvre et qui sera décrite dans les paragraphes suivants. Les poids α représentent une sorte de «confiance» attribuée à chaque hypothèse faible, ainsi la multiplication est une pondération de la décision finale de chaque arbre. Les résultats des pondérations, c'est à dire les sorties des multiplieurs, sont additionnés en utilisant une architecture pyramidale afin d'obtenir la décision finale de la classification, laquelle est la somme des décisions finales de tous les modules d'hypothèses faibles. Le temps de latence de ce module est $2 \times (n_t + 1) + \lceil \log_2 (T) \rceil + 1$ tops de l'horloge pixel.

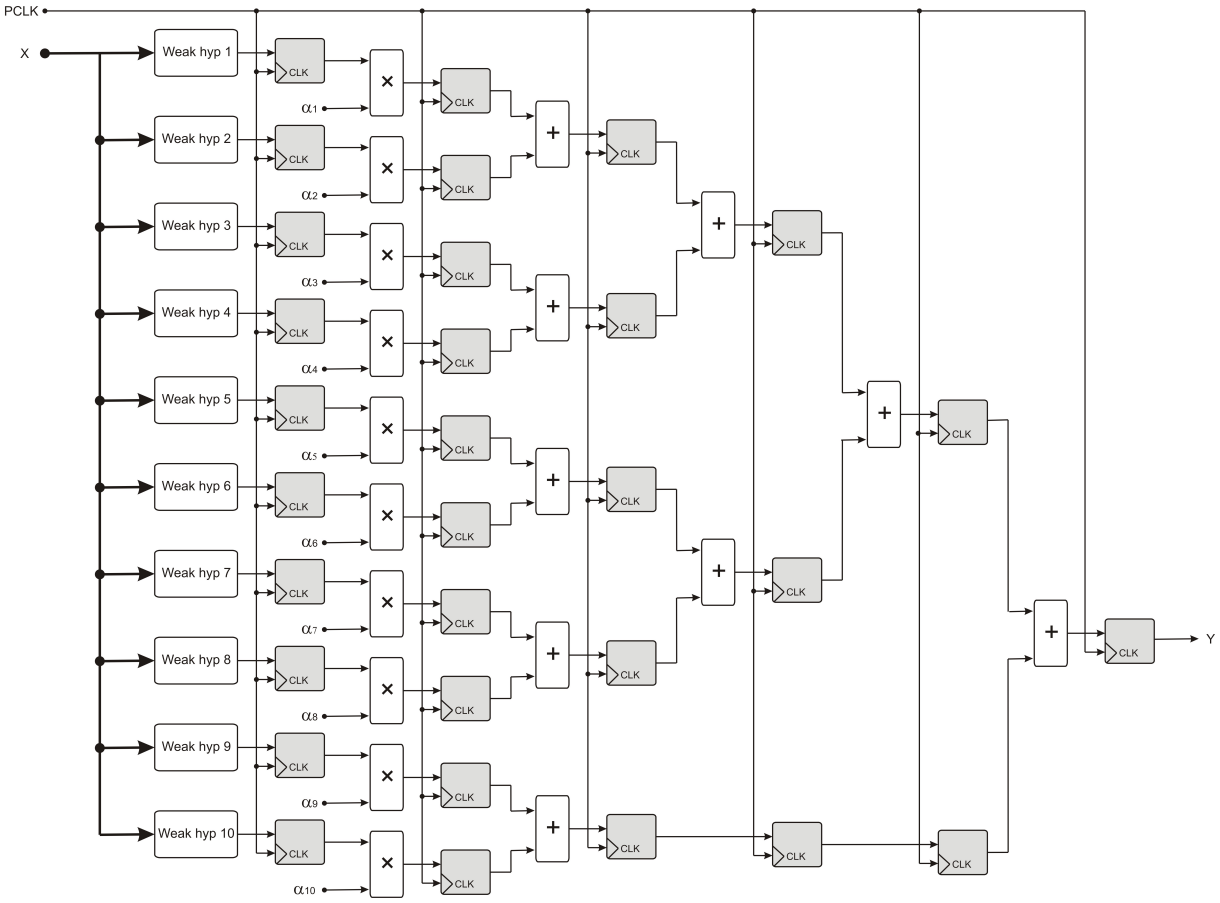


FIGURE 5.14 – L'architecture du module de classification basé sur l'algorithme AdaBoost.

Apprentissage et génération automatique des modules de classification

Le module de classification change constamment en fonction de l'application, la performance requise et les ressources disponibles sur le dispositif matériel. En conséquence chaque fois qu'on aura besoin de générer un nouveau module de classification, l'architecture de base sera conservée et seulement les paramètres de configuration seront mis à jour. Cela revient à ajuster $\left(2^{n_t+2} - 2\right) \times T = 126 \times T$ paramètres, où T représente le nombre d'hypothèses faibles utilisées dans l'algorithme AdaBoost. Pour optimiser le processus de création des modules de classification et pour éliminer les erreurs humaines possibles, nous avons mis en place une méthode d'apprentissage et de génération automatique des modules de

classification.

La figure 5.15 montre le diagramme de la méthode mise en place pour automatiser l'apprentissage et la génération des modules de classification. Cette méthode exploite en entrée, soit une base d'images de l'environnement, lesquelles seront ensuite utilisées pour obtenir les attributs de la couleur et de la texture à l'aide de l'outil de simulation ModelSim, soit les attributs eux-mêmes calculés à partir d'un FPGA et d'une caméra. Cette derniere option nécessite un outil de communication pour pouvoir télécharger les attributs depuis la mémoire du FPGA vers un fichier.

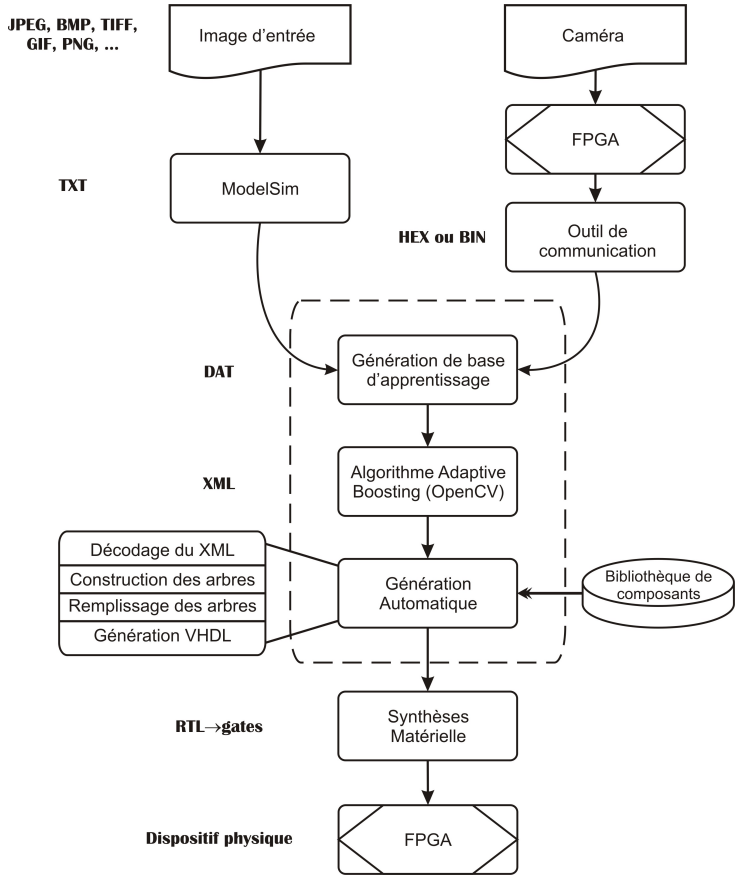


FIGURE 5.15 – Diagramme de la méthode d'apprentissage et de génération automatique des modules de classification fondées sur l'algorithme AdaBoost..

Les fichiers de la base des images ou des attributs, obtenus depuis l'outil de simulation ModelSim ou l'outil de communication, sont utilisés pour créer la base d'apprentissage. Ce processus est réalisé par l'utilisateur de façon manuelle en utilisant l'interface graphique développée à cet effet. La figure 5.16 montre cette interface graphique : à gauche l'image en cours de traitement, à droite en haut une partie de cette image qui a été sélectionnée pour être classée comme obstacle ou sol, au-dessous de cette partie de l'image deux boutons avec lesquels on choisit la classe d'appartenance de l'échantillon ou partie de l'image. Au bas des deux boutons, deux indicateurs qui donnent le nombre d'échantillons dans chacune des classes et contribuent à l'équilibre entre les classes contenues dans la base d'apprentissage. Au-dessous des indicateurs des échantillons, un bouton pour effacer la base d'apprentissage et recommencer le processus de sélection des échantillons. Plus bas se trouve le bouton pour enregistrer la base d'apprentissage et pour poursuivre la méthode. Au-dessous de ce bouton, un indicateur montre le chemin d'accès de la base d'images, le bouton du bas est utilisé pour charger une nouvelle base d'images et l'indicateur

à droite donne le nombre d'attributs utilisés dans la base d'images.

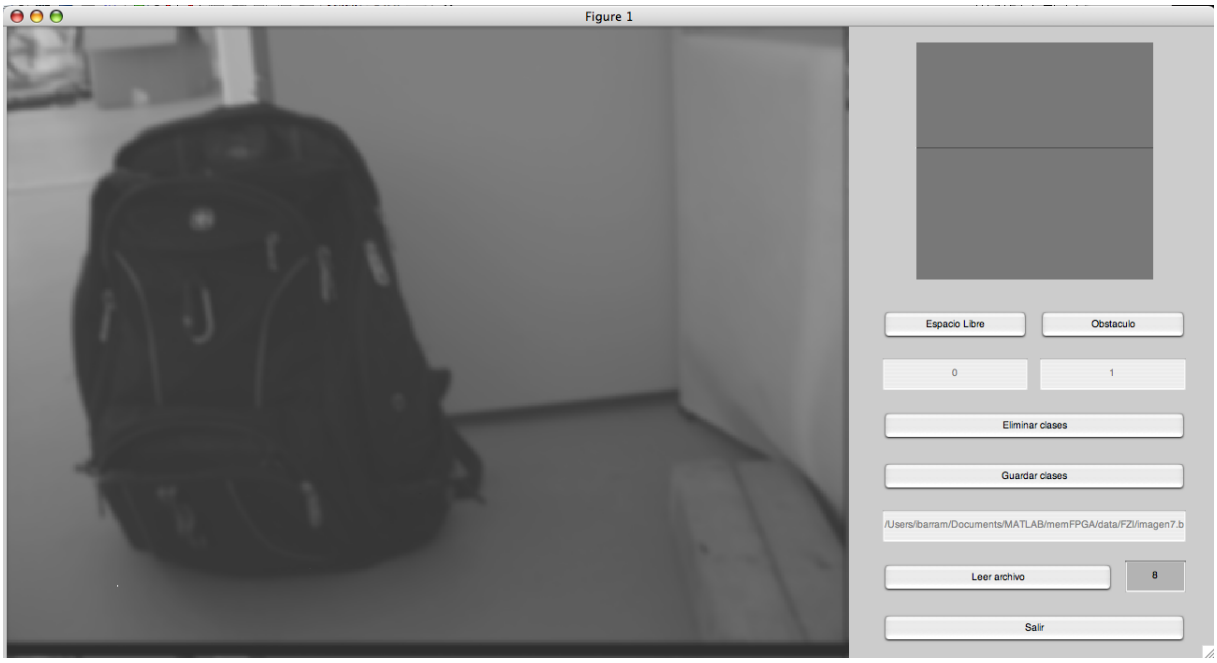


FIGURE 5.16 – L'interface graphique de l'outil pour la création de la base d'apprentissage.

Une fois sélectionnés les échantillons, on les enregistre dans la base d'apprentissage. L'outil génère un fichier .DAT qui est transmis à l'outil d'apprentissage développé en langage C et utilise la librairie de vision par ordinateur appelée OpenCV. Cet outil d'apprentissage lit le fichier et construit un classificateur basé sur l'algorithme AdaBoost. Celui-ci est ensuite enregistré dans un fichier XML puis ré-exploité par l'outil pour générer le fichier VHDL à partir de notre bibliothèque de composants. Le processus de génération automatique du fichier VHDL est composé de cinq étapes :

1. **Le décodage du fichier XML.** La première étape du processus de génération automatique des modules de classification est le décodage du fichier XML qui contient toutes les informations du classificateur. Ce processus crée une représentation vectorielle pour chacun des poids de chaque hypothèse faible tout en effectuant la lecture des seuils et des indices des composants de comparaison dans les arbres de décision.
2. **Construction des arbres.** Dans cette deuxième étape, on reprend les seuils et les indices des composants de comparaison pour générer une représentation hiérarchisée de chacun des arbres de décision. La figure 5.17 montre un exemple d'une représentation hiérarchique d'un arbre de décision obtenu à partir de l'outil d'apprentissage via le fichier XML. Dans cette figure, les ovales représentent les branches de l'arbre tandis que les trapèzes représentent les feuilles de chaque arbre. Les seuils de chaque branche sont représentés par le vecteur c, tandis que le bus d'entrée dans le temps i est représenté par la matrice x, où le vecteur j contient les indices des composantes à comparer dans ce bus pour chaque branche. Les valeurs *cl0* et *cl1* correspondent aux classes zéro ou un. Nous devons nous rappeler que l'algorithme d'apprentissage crée un arbre de décision avec un niveau de profondeur maximale de cinq, mais il peut fournir un niveau de profondeur inférieur si celui-ci est suffisant pour séparer les deux classes.
3. **Remplissage des arbres.** Dans cette étape, les arbres sont complets afin que les branches qui ont un niveau de profondeur inférieur à cinq soient prolongées. Ce processus est effectué afin d'homogénéiser les hypothèses faibles et donc pouvoir définir tous les arbres de décision avec un seul

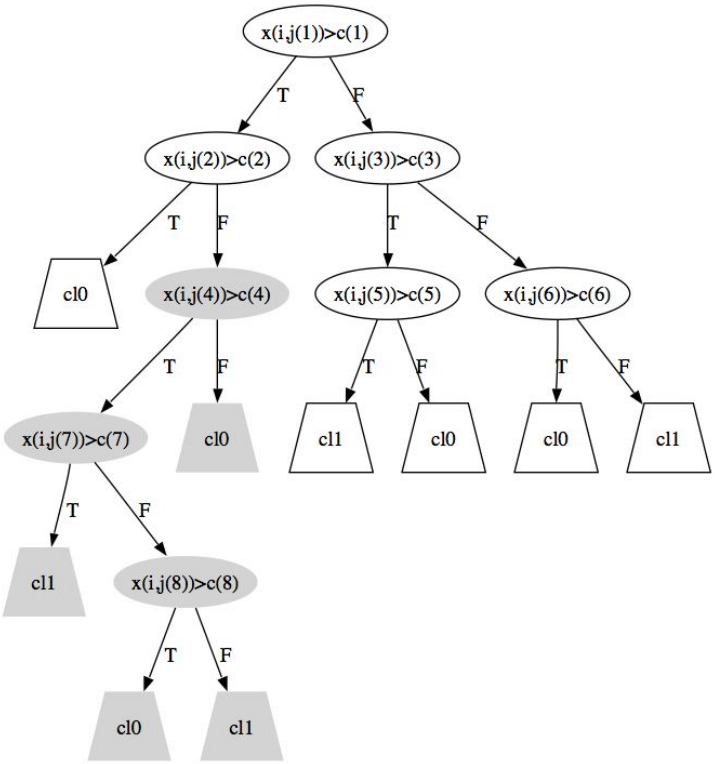


FIGURE 5.17 – Exemple d'une représentation hiérarchique d'un arbre de décision généré automatiquement par OpenCV. Cet arbre de décision est utilisé comme hypothèse faible pour l'algorithme Adaptive Boosting sur logiciel.

module paramétrable. Le processus de création du module de plus haut niveau dans la hiérarchie de la classification se réduit à la configuration de chacun des modules et à la combinaison de leurs réponses. A titre d'exemple, les figures 5.18 et 5.17 montrent en gris le remplissage des différentes branches. Dans la figure 5.18 les branches et les feuilles en couleur jaune représentent celles qui ont été ajoutées à l'arbre de décision original. On prend à chaque fois la première composante dans le bus d'entrée pour assurer son existence : si la classe dans la feuille avant le remplissage avait une valeur zéro alors la valeur de comparaison est fixée à $+\inf$ pour s'assurer que le chemin de droite dans les branches sera toujours sélectionné. Dans le cas contraire, c'est-à-dire si la classe dans la feuille de l'arbre original avait une valeur de un alors le seuil de comparaison sera fixé à $-\inf$. Enfin, les classes dans les feuilles correspondent aux classes des feuilles dans l'arbre d'origine avant d'être rempli.

4. **Normalisation des paramètres.** Une fois rempli l'arbre de décision, il est nécessaire de normaliser les différents paramètres, à l'exception des indices des composantes et des seuils car ceux-ci sont liés à la taille et la gamme du vecteur d'entrée et donc du bus. Toutefois, dans le cas du vecteur de pondération α dans le module AdaBoost de plus haut niveau dans la classification, il est nécessaire de prendre en compte les valeurs maximale et minimale de la somme des poids. L'équation 5.11 est utilisée pour normaliser le poids $\hat{\alpha}_i$ obtenu à partir du décodage du fichier XML. Ce poids est divisé par le double de la somme absolue des poids, ce qui correspond à une gamme située entre les valeurs maximale et minimale. Ce résultat est multiplié par un facteur d'échelle qui permet de normaliser sa valeur à n_b bits et dans laquelle le premier bit correspond au signe indiquant la classe à laquelle appartient le pixel courant et les bits restants représentant la probabilité d'appartenance

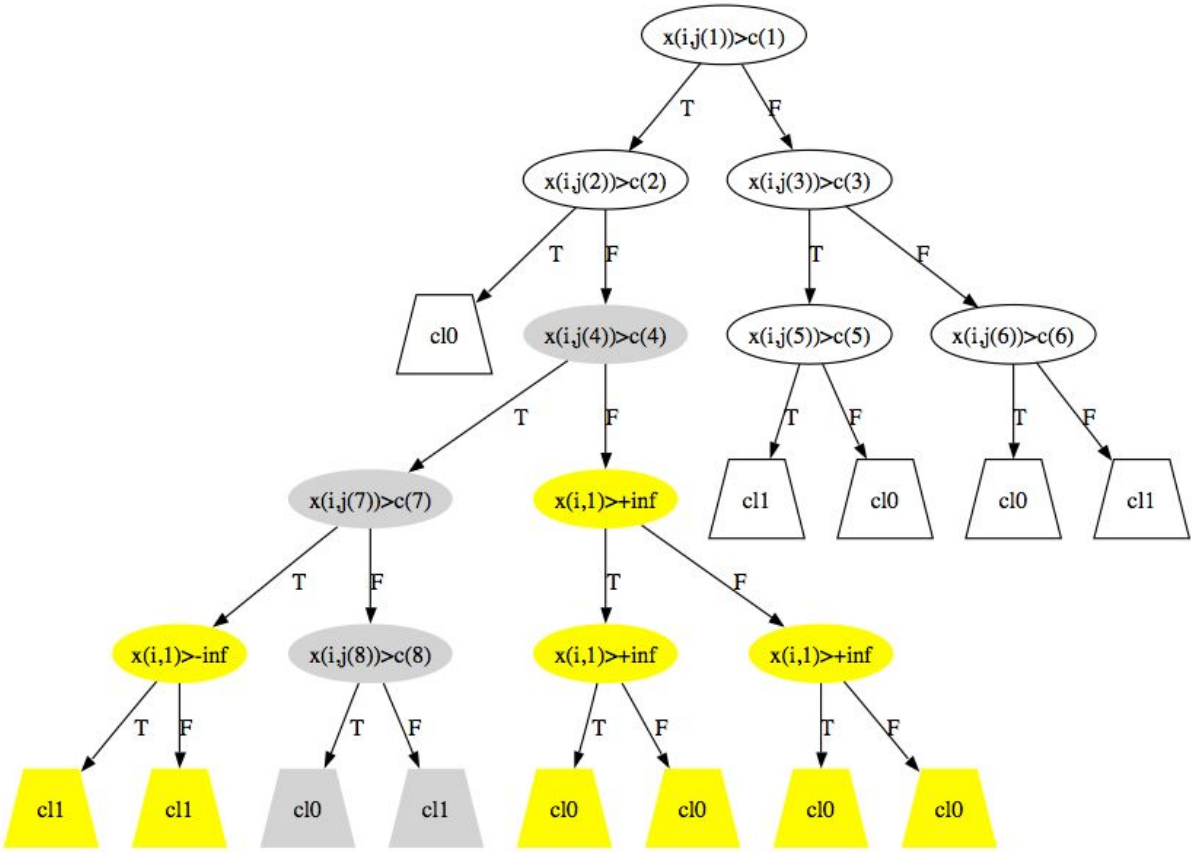


FIGURE 5.18 – Exemple d'une représentation hiérarchique d'un arbre de décision généré automatiquement par la méthode d'apprentissage mise en place. Cet arbre de décision est utilisé comme hypothèse faible pour le portage sur matériel de l'algorithme Adaptive Boosting.

à la classe attribuée.

$$\alpha_i = 2^{n_b-2}\frac{\hat{\alpha}_i}{2\sum|\hat{\alpha}|} \tag{5.11}$$

5. **Génération du fichier VHDL.** La dernière étape est la génération du module décrit en langage VHDL. Pour cela on s'appuie sur l'architecture présentée dans la figure 5.14 : chaque module correspondant à une hypothèse faible est configuré en utilisant les indices des composantes et les valeurs de seuils issues de l'arbre pré-rempli. Les sorties des modules d'hypothèse faible sont multipliées par le poids normalisé correspondant. Les sorties des multiplications sont ensuite additionnées successivement entre elles dans une structure pyramidale fournissant le résultat final de la classification. Ce module est directement intégré dans notre architecture globale de détection d'obstacles.

5.2 Evaluation globale et analyse comparative des architectures

Dans cette section nous présentons deux évaluations : la première est une analyse comparative entre les différentes architectures d'analyse de texture trouvées dans la littérature et la nôtre. La deuxième

évaluation est une analyse comparative entre notre module de classification fondé sur l'apparence et deux modules similaires issus de la littérature.

La première architecture à comparer est l'architecture d'analyse de texture. Celle-ci a été implémentée sur un circuit FPGA de la famille Cyclone III EP3C120F780C8 de chez Altera. Les principaux résultats de la synthèse du dispositif sur FPGA indiquent une consommation de $62,694$ fonctions combinatoires et de $45,708$ registres logiques dédiés ; les deux combinés représentent $73,021$ éléments logiques. La mémoire requise est de $737,152$ bits et le nombre de multiplieurs embarqués est de 480. Les éléments logiques utilisent 61% de la capacité totale du circuit alors que la taille de la mémoire et les multiplieurs représentent 19% et 83% respectivement. Ces résultats dépendent directement de la taille de l'image, du nombre de déplacements, de la distance et de la direction des déplacements et de la taille de la fenêtre de traitement utilisés pour obtenir les attributs de texture. Pour une image de 640×480 pixels, notre architecture fournit avec succès 30 images denses d'attributs de texture par seconde.

Tout d'abord, nous faisons un comparatif entre les performances de l'algorithme implanté sur notre architecture matérielle et celui implémenté de manière logicielle sur un PC MacBook Pro équipé d'un processeur Intel Core 2 Duo à 2.33 GHz, 4 Mb de cache niveau 2 et 2 Gb de mémoire RAM. Les résultats sont présentés dans le tableau 5.1 pour différentes tailles d'image et nombres de déplacements (première et deuxième colonnes, respectivement). La troisième colonne indique le temps de traitement lié à l'application pour laquelle la différence de temps calcul est perceptible. Dans le cas de l'implémentation sur FPGA, le parallélisme et la flexibilité permettent d'atteindre des temps de traitement très inférieurs avec un facteur d'accélération élevé par rapport à la solution PC. On peut remarquer que le facteur d'accélération est directement proportionnel au nombre de déplacements. Ces résultats soulignent également que le nombre de déplacements n'affecte pas le temps de traitement sur le circuit FPGA car chaque déplacement est calculé sous forme parallèle.

TABLE 5.1 – Tableau comparatif de la performance pour différentes implémentations.

Taille de l'image (pixels)	Nombre de déplacements	Temps de traitement		Facteur d'accélération
		PC	FPGA	
320×240	1	$0.30s$	$7.68ms$	39.06
640×480	1	$1.21s$	$30.72ms$	39.38
640×480	4	$4.68s$	$30.72ms$	152.3
640×480	8	$9.22s$	$30.72ms$	300.1
640×480	16	$18.4s$	$30.72ms$	600.9
1024×768	1	$2.72s$	$78.64ms$	34.58

Ensuite, la mise en œuvre proposée sur FPGA a été comparée à quatre autres architectures trouvées dans la littérature. Les principales caractéristiques de ces architectures sont présentées dans le tableau 5.5. Les paramètres de configuration et les performances sont présentées dans la première colonne et les suivantes indiquent les valeurs associées pour les quatre types architectures. Ces architectures ont été présentées par différents auteurs (on trouvera dans les références [85], [132], [102] et [127], plus de détails techniques sur ces mises en œuvre). Ces quatre architectures, toutes mises en œuvre sur FPGA, calculent l'image des attributs de texture en utilisant soit des matrices de co-occurrence en niveaux de gris (GLCM), soit des histogrammes de co-occurrence en niveaux de gris (GLCH). Notre architecture peut être comparée directement avec l'architecture de Ibarra-Pico bien que celle-ci soit à base de FPGA et DSP parce que les deux utilisent des histogrammes pour calculer les attributs de la texture. Cependant cette dernière utilise directement l'image d'entrée ce qui réduit son efficacité dans le calcul des attributs de la texture comparativement à notre architecture qui présente un meilleur compromis entre perfor-

mance, nombre des attributs de la texture et nombre de niveaux de gris. Bien que les architectures de Tahir, Maroulis et Siéler utilisent des matrices de co-occurrence pour calculer les attributs de la texture, ces architectures traitent l'image de sorte qu'il n'existe pas de chevauchement entre les fenêtres de traitement. Cela permet de réduire le temps de traitement, les ressources utilisées mais aussi les performances de la classification et de la détection. La mesure de la performance de traitement *pp*, dans l'équation 5.12, représente le flux d'informations traitées par l'architecture et sera utilisée pour les comparer entre elles. Cette mesure dépend directement de l'information d'entrée, des paramètres de l'architecture et s'exprime en octets par seconde (bps). Notre architecture présente une plus grande valeur de *pp*, bien que l'architecture de Maroulis ait la performance la plus élevée, mais cette dernière calcule moins d'attributs de texture et une taille d'images d'entrée plus petite. Contrairement aux architectures de Tahir, Maroulis et Siéler, la nôtre utilise un plus grand nombre de niveaux de gris et calcule les attributs de la texture de sorte que les fenêtres de traitement se chevauchent. Enfin, les architectures de Ibarra-Pico, Tahir et Maroulis requièrent au moins une mémoire externe et interne, celle de Siéler requiert seulement une mémoire interne comme la nôtre mais de taille supérieure.

$$pp = \begin{cases} U \times V \times B \times K \times L \times log2\left(N_g^{\frac{1}{8}}\right) \times f \times d \times p & si\ chevauchement \\ U \times V \times B \times log2\left(N_g^{\frac{1}{8}}\right) \times f \times d \times p & si\ pas - chevauchement \end{cases} \tag{5.12}$$

Nous avons procédé à une deuxième évaluation de notre architecture de classification basée sur les attributs de la couleur et de la texture [80]. Celle-ci a été portée sur un FPGA de la famille Cyclone II EP2C70F896C6 de chez Altera. Les principaux résultats de la synthèse sont résumés dans la table 5.2. Les ressources requises sont principalement fonction de la taille de l'image, du nombre de déplacements, de la taille de la fenêtre de traitement utilisée par le module d'analyse des attributs de la texture et du nombre d'hypothèses faibles utilisées par le module de classification. Dans cette architecture, nous utilisons une taille d'image de 640×480 pixels avec 1 seul déplacement et une fenêtre de traitement de 17×15 pixels. Enfin, nous comparons les performances et les ressources consommées pour 10, 20 et 30 hypothèses faibles dans le module de classification. Avec ces paramètres et un rythme d'horloge de 50 MHz, notre architecture classe 30 images denses par seconde.

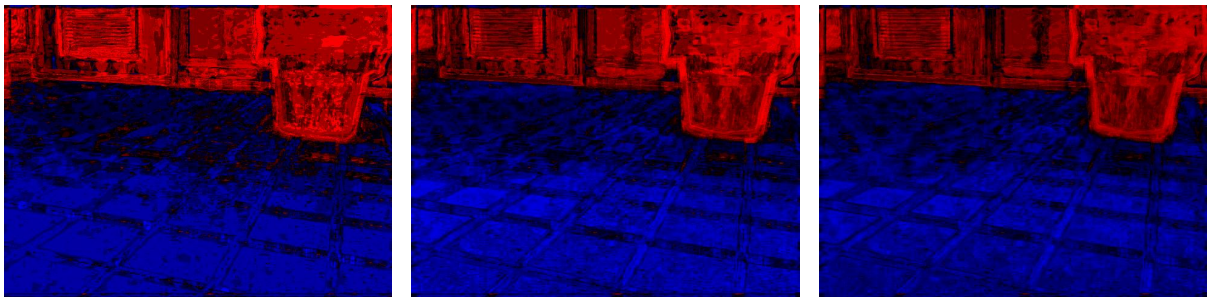
Dans la table 5.2, la première colonne représente les différents types de ressources consommées par le module. La deuxième colonne indique les statistiques des ressources consommées par le module de transformation de la couleur. Du fait que les coefficients dans la matrice de transformation sont constants, ce module n'utilise que des éléments logiques. Contrairement au module de la transformation de la couleur, le module d'analyse de la texture utilise la mémoire pour stocker les pixels nécessaires au calcul des sommes et différences des images et au calcul des attributs de la texture. Nous avons notablement réduit les ressources de mémoire compte tenu du fait que les fenêtres de traitement utilisées sont communes aux différents attributs. Ce module utilise les multiplieurs embarqués pour calculer les attributs de contraste et de variance. Les trois autres colonnes dans la table 5.2 se réfèrent au module de classification basé sur l'algorithme AdaBoost pour un nombre de 10, 20 et 30 hypothèses faibles. Le nombre d'éléments logiques est directement proportionnel au nombre d'hypothèses faibles. A contrario, le nombre de multiplieurs embarqués est obtenu en soustrayant le nombre d'hypothèses faibles (poids total) au nombre de poids qui sont multiples de deux ; cela vient du fait que dans ce cas, le multiplieur embarqué est remplacé par une opération de décalage.

La figure 5.19 montre les images résultant du processus de classification pour 10, 20 et 30 hypothèses faibles. La probabilité de la détection positive est représentée en couleur bleu, ce qui, dans le cas de ces images, représente le sol ou l'espace libre. Quand le nombre de classificateurs augmente, on obtient une meilleure performance dans le classement, cependant, cela se traduit par une augmentation des ressources

TABLE 5.2 – Table des statistiques de ressources consommées par l'architecture de détection d'obstacles.

	Attributs de la couleur	Attributs de la texture	Classification (# hypothèses faibles)		
			10	20	30
Eléments Logiques	716 (1%)	19,766 (29%)	7,683 (11%)	14,078 (21%)	19,401 (28%)
Fonctions Comb.	643 (1%)	15,676 (23%)	5,676 (8%)	11,615 (17%)	16,826 (25%)
Registres Dédiés	280 (1%)	12,591 (19%)	5,229 (8%)	8,210 (12%)	10,543 (15%)
T. de la mémoire	–	379 Kb (34%)	–	–	–
Multiplieurs Emb.	–	120 (40%)	36 (12%)	68 (23%)	64 (21%)

consommées et du temps de latence. Ce fait est facile à vérifier en observant les zones sombres et rouges autour de la poubelle sur l'image de gauche, lesquelles correspondent à des zones mal classées, puis progressivement ces zones s'éclaircissent sur les images du milieu et de droite.



(a) (b) (c)

FIGURE 5.19 – Images résultat de la classification des obstacles fondée sur l'apparence pour différents nombres d'hypothèses faibles.

La figure 5.20 montre trois situations différentes utilisées pour évaluer l'architecture proposée. Ces résultats sont obtenus en utilisant les deux composantes chromatiques a et b dans l'espace de couleur CIE-L*ab, six attributs de la texture calculés pour une taille de fenêtre de 17×15 pixels et 30 hypothèses faibles dans l'algorithme AdaBoost. La première colonne montre les images d'origine dans l'espace de couleur RGB. La colonne du milieu montre les images résultant du processus de classification : la couleur bleue représente la probabilité de détection positive, la couleur rouge est la probabilité de détection négative et la couleur sombre est la probabilité proche de zéro. Dans le but d'obtenir une image plus simple à interpréter, un processus de seuillage est appliqué à l'image de probabilité pour obtenir l'image binaire de la colonne de gauche. Nous mettons en évidence la complexité du test avec la bouteille d'eau, parce que cet obstacle reflète une partie des attributs de la couleur et de la texture. Ce cas représente une situation très difficile à traiter pour les algorithmes couramment fondés sur la couleur ou la texture, mais une combinaison de deux stratégies différentes permet d'obtenir un bon niveau de détection de ce type d'objet.

En outre nous comparons les performances entre les algorithmes portés sur matériel et les mêmes mis en œuvre sur un PC MacBook Pro équipé d'un processeur Intel Core 2 Duo à 2.33 GHz, 4 Mb de cache niveau 2 et 2 Gb de mémoire RAM. Les résultats de la performance entre les deux solutions sont présentées dans la table 5.3, pour différentes tailles d'image et de nombre d'hypothèses faibles (la première et la deuxième colonne, respectivement). La troisième colonne indique le temps de traitement pour chaque mise en œuvre, la différence en unités de temps est perceptible. Dans le cas des FPGAs, le traitement en parallèle et la flexibilité du calcul permettent d'atteindre des temps de traitement inférieurs. La dernière

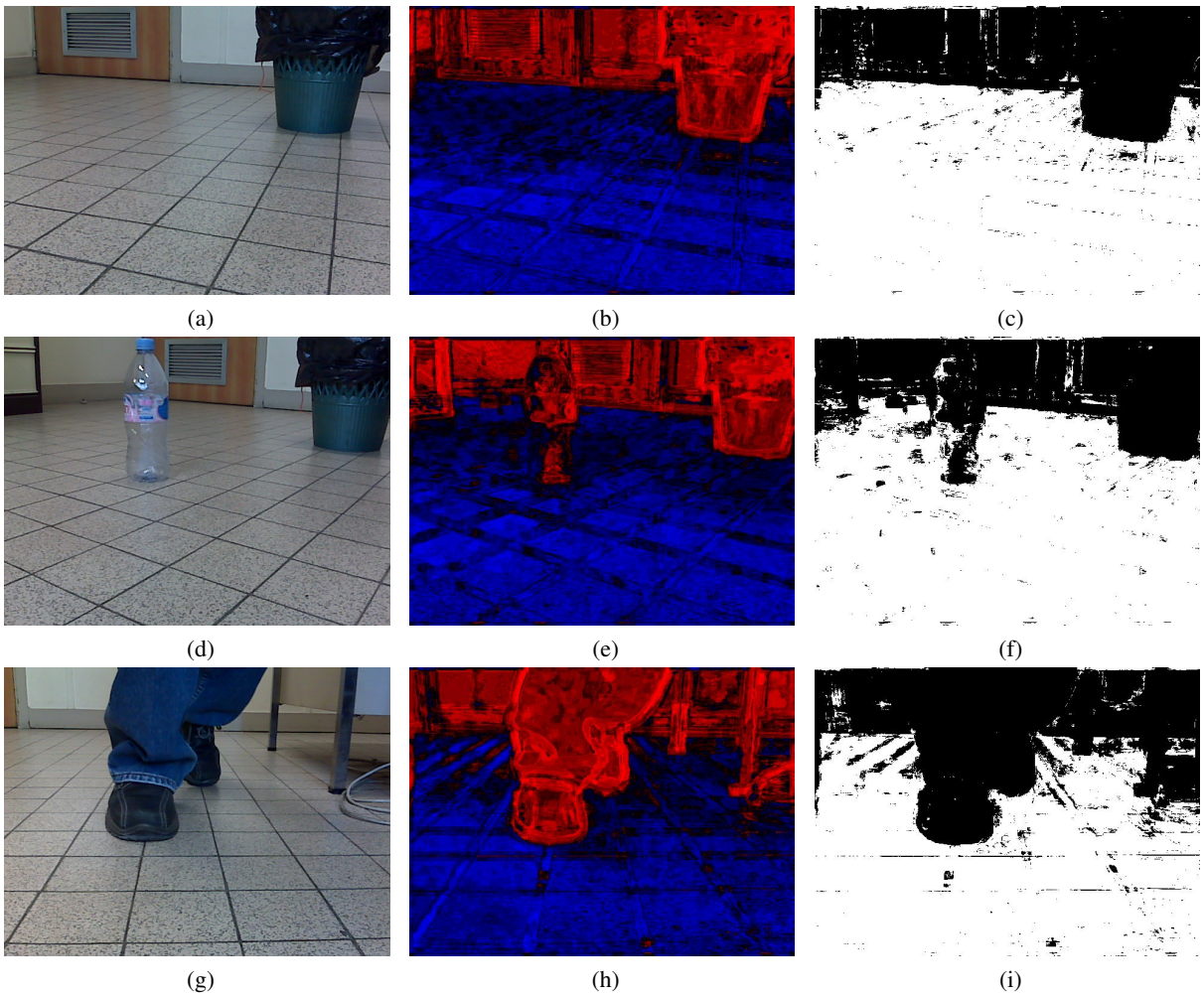


FIGURE 5.20 – Images originales et images résultant de la classification pour 30 hypothèses faibles dans l'algorithme AdaBoost, (a), (d) et (g) images originales dans l'espace couleur RGB, (b), (e) et (h) images de probabilité de détection obtenues à partir de l'architecture de classification et (c), (f) et (i) images seuillées obtenues à partir de l'architecture de classification.

colonne donne un facteur d'accélération élevé par rapport à la solution PC. On peut noter que ce facteur est directement proportionnel au nombre d'hypothèses faibles. Ces résultats soulignent également que le nombre d'hypothèses faibles n'affecte pas le temps de traitement dans le FPGA parce que celles-ci sont calculées simultanément contrairement à la solution PC pour lequel la durée du traitement dépend directement de leur nombre. Cependant contrairement aux FPGA, la solution PC est facile et rapide à mettre en oeuvre.

Enfin, l'architecture de classification proposée sur FPGA a été comparée à deux autres trouvées dans la littérature. La table 5.4 montre le bilan de l'implémentation de notre architecture et des deux autres. Notre architecture utilise à la fois les attributs de la couleur et de la texture, les deux autres utilisent seulement les attributs de la texture. Notre architecture et celle de Lopez-Estrada [99] utilisent la classification intégrée sur le FPGA, ce qui augmente leur performance. Cependant, l'arbre de décision utilisé dans l'architecture de Lopez-Estrada est un classificateur moins robuste car il utilise un nombre plus limité d'attributs, ce qui réduit son champ d'utilisation. La surface, dans notre architecture, ne peut pas être directement comparée aux architectures de Tahir [132] et de Lopez-Estrada lesquelles sont implé-

TABLE 5.3 – Table comparative de la performance pour différentes implémentations.

Taille de l'image (pixels)	Nombre de hypothèses faibles	Temps de traitement		Facteur de d'accélération
		PC	FPGA	
320×240	30	$617ms$	$7.68ms$	80.37
640×480	10	$1.52s$	$30.72ms$	49.66
640×480	20	$1.96s$	$30.72ms$	63.83
640×480	30	$2.45s$	$30.72ms$	79.96
640×480	60	$3.80s$	$30.72ms$	123.9
1024×768	30	$6.43s$	$78.64ms$	81.76

TABLE 5.4 – Table comparative de la performance des algorithmes de classification basés sur l'apparence pour différentes architectures portées sur FPGA.

	Our Design	Tahir [132]	Lopez-Estrada [99]
Attributs de la couleur	CIE-L*ab	–	–
Attributs de la texture	SDHs	GLCM	GLCM
Algorithme de classification	AdaBoost	LOO (PC)	Decision Tree
Taille de l'image	640×480	512×512	256×256
Taille de la fenêtre de traitement	17×15	128×128	256×256
Nombre de niveaux de gris	256	32	256
Nombre d'attributs	8	7	3
Performance	30 fps	26 fps	–
Surface	34, 560 LE	20, 544 S	277 S
Taille de la mémoire interne	379 Kb	400 Kb	–
Taille de la mémoire externe	–	8 Mb	–

mentées sur des FPGAs de chez Xilinx. Toutefois, il est important de souligner que, contrairement à notre architecture, les deux autres traitent l'image de sorte qu'il n'y ait pas de chevauchement entre les fenêtres de traitement, c'est à dire qu'elles ne donnent pas une classe pour chaque pixel dans l'image, mais donnent une classe pour chaque ensemble de pixels dans la fenêtre de traitement. Cela diminue le nombre d'opérations et par conséquent les ressources nécessaires et augmente la performance. En fait, l'architecture de Lopez-Estrada donne une seule classe pour toute l'image, c'est à dire qu'elle détecte si la classe existe dans l'image ou pas mais elle n'indique pas dans quelle partie de celle-ci se situe l'objet, elle en détecte seulement l'existence.

5.3 Système intégré de vision multi-caméras pour la détection des obstacles

Dans cette section, nous présentons l'extension de notre architecture de détection des obstacles à un système multi-caméras. Tout d'abord, nous avons développé un outil de simulation pour configurer plus rapidement les différents paramètres des micro-caméras et optiques, la figure 5.21 présente l'interface graphique de cet outil. Les caméras sont disposées symétriquement autour du robot. Les contrôles sur la

gauche en haut configurent la taille du robot dans sa longueur, sa hauteur et sa largeur. Au-dessous, l'utilisateur dispose d'une fenêtre pour configurer les paramètres intrinsèques de la caméra et de l'optique : taille de la matrice, taille du pixel, distance focale, les coordonnées du point principal et résolution. S'il connaît sa caméra, l'utilisateur met là directement les paramètres correspondants : ici figirent les paramètres d'une micro-caméra SCT 1/3" équipée avec un objectif 3.2mm. Les fenêtres sur la droite en haut permettent de configurer les positions des caméras sur le robot, sachant que c'est une ceinture, donc même élévation pour toutes les caméras et disposition régulière autour du robot. Enfin, ce programme affiche au centre de la fenêtre, la simulation des champs de vue des caméras : robot en bleu, le sol en vert, caméras en rouge et champs de vue sur le sol de chaque caméra selon les paramètres de configuration en jaune.

Une configuration optimale des caméras doit réduire au minimum les zones aveugles autour du robot et doit optimiser la taille minimale et la distance maximale pour un obstacle détecté. Il est possible d'éviter tout chevauchement entre les champs de vue, ou de tolérer de telles duplications, afin d'accroître la confiance des mesures des capteurs par la redondance des informations. Ici avec un robot circulaire, les zones du sol perçues par deux caméras sont très réduites ; mais si le robot a une forme rectangulaire (comme les ATRV par exemple), alors on pourrait exploiter la stéréovision dans le champ commun des caméras affectées à un même côté.

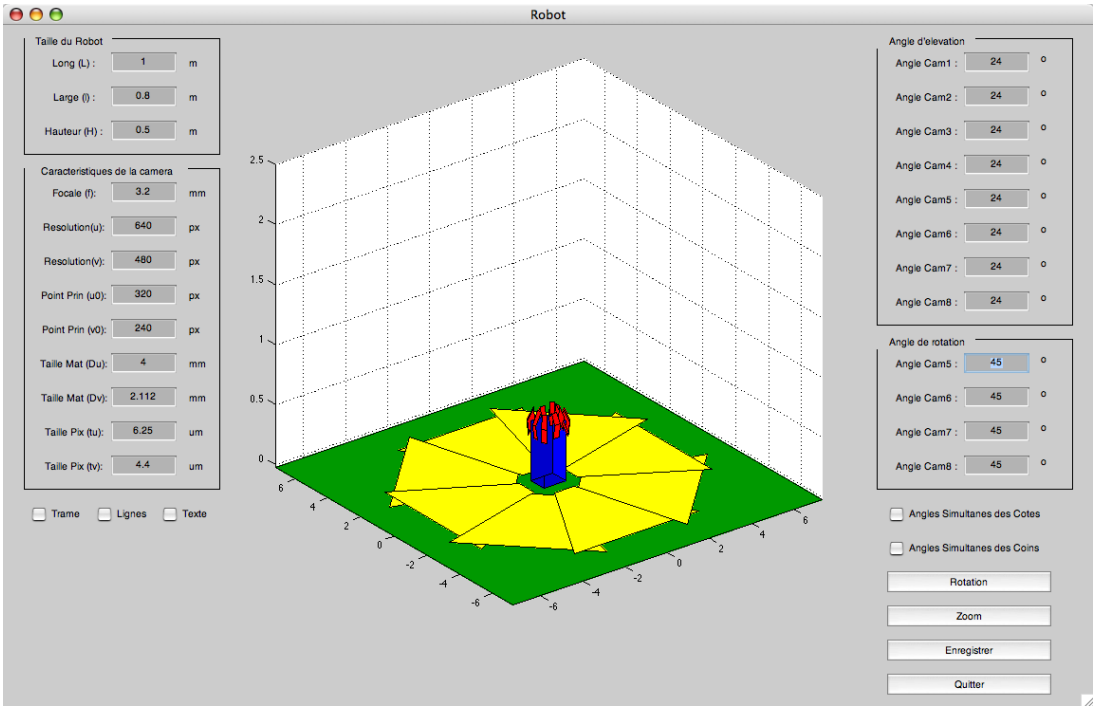


FIGURE 5.21 – Interface graphique de l'outil de simulation pour le système de vision multi-caméras pour la détection des obstacles.

5.3.1 Sélection des micro-caméras

Ensuite, plusieurs micro-caméras ont été analysées afin de sélectionner le produit que nous allions intégrer dans le système final. Plusieurs contraintes doivent être remplies : les micro-caméras devraient pouvoir être intégrées par nous-mêmes avec une compacité optimale, la flexibilité et la longueur des câbles entre les micro-caméras et l'unité de traitement devaient permettre d'évaluer le système sur des

robots de grande taille, ou même d 'exploiter la même configuration pour d'autres applications telles que la vidéo-surveillance.

Nous avons comparé différentes solutions : la micro-caméra couleur STMicroelectronics VS6724 présentait un bon compromis, mais nous n'en avions qu'une (donnée par SCT), et il est impossible d'acheter ces caméras en une faible quantité. Le kit caméra TERASIC est très couramment exploité avec des kits Altera, mais il ne dispose que d'une connexion parallèle, donc la nappe caméra/unité de traitement doit être très courte (20cm). La famille des micro-caméras Micron-Aptina MT9Vxxx présente de nombreux avantages, sauf qu'il faut développer un interface dédié en LVDS. Finalement, une procédure en deux étapes a été utilisée :

- la contrainte de longueur de câble a été d'abord relâchée afin de permettre des progrès sur la conception de l'architecture ; les caméras avec des liens parallèles (c'est à dire des câbles très courts) sont actuellement utilisés. Il s'agit du kit TERASIC doté d'un capteur de 5 Mega-pixels et d'une connexion parallèle avec les connecteurs GPIO des kits de développement.
- les micro-caméras du prototype final sont en cours d'élaboration : le choix s'est fixé sur la micro-caméra CMOS couleur MT9V034 Aptina qui a une résolution Wide VGA 752×480 et peut aller jusqu'à 60 images par seconde avec une connexion LVDS qui peut être branchée à une distance maximale de 8 m. Le capteur MT9V034 dispose d'une Global shutter-TrueSNAP une option très utile pour l'acquisition d'image sur des scènes dynamiques ou pour les caméras mobiles.

5.3.2 Architecture pour la détection de obstacles

La première validation globale d'un système de 8-caméras montées sur un robot, sera fait en utilisant un kit de développement Stratix III de chez Altera. Le FPGA est un Stratix III EP3SL150F1152 de haute performance avec environ $140,000$ éléments logiques et 384 multiplieurs embarqués (18×18). L'environnement de développement est Quartus II pour le projet de haut niveau et l'outil SOPC pour toute l'architecture d'interconnexion. Nous utilisons NIOS IDE pour les parties laissées en logiciel, quelques tâches comme la gestion des registres des caméras via I2C et le serveur socket réseau. La configuration matérielle actuelle utilise 4 caméras connectées en parallèle par les ports HSMC à l'aide de deux adaptateurs Terasic HSMC à GPIO.

Le kit de développement Stratix III a également été choisi pour la diversité des mémoires disponibles. Les mémoires actuellement utilisé par l'architecture courante sont les suivants :

- La mémoire PSRAM a deux modules avec une largeur d'interface de 32 bits ; elle travaille en mode synchrone à 100 MHz. Elle est utilisée en mode partagé par tous les modules liés aux caméras, pour écrire les images.
- Deux mémoires DDR2 indépendantes avec une interface de 8 bit et une capacité de 16 Mb chacune, en travaillant à 200 MHz délivrant chacun un débit de 100 Mega mots par seconde pour une taille de 32 bits par mot. Le premier bloc DDR2 est utilisé comme mémoire du programme du processeur NIOS. Bien que la taille de l'application du socket serveur est à peu près quelques centaines de kilo-octets, il est déjà hors de portée pour une utilisation de la mémoire interne du FPGA, qui est réservée de toute façon, aux données et aux caches du programme. Le deuxième bloc DDR2 est uniquement exploité pour le réseau à la fois pour la transmission et la réception du SGDMA (Scatter Gather Direct Memory Access), connectés avec la Speed Triple Ethernet IP.

La communication

L'unité de traitement sur le kit StratixIII doit acquérir de manière synchrone à 30Hz, un flux de 8 séquences vidéo simultanément, puis doit lancer en parallèle les traitements sur 8 images à chaque instant t. Toutes les informations sont fusionnées de façon probabiliste, dans une grille d'occupation

robot-centrée, une grille de 100×100, où la probabilité d'occupation est calculée et mise à jour pour chaque cellule également à 30Hz. Dans l'état actuel, il est prévu que cette grille soit envoyée au système hôte, c'est à dire l'unité de locomotion pour un robot autonome ou une unité d'affichage pour un véhicule à moteur. La taille de la grille est d'environ 200 Kb. En outre, pour les besoins de débogage et de démonstration, les images originales, corrigées ou classés (en résolution VGA et sans la couleur, 8 bits par pixel, soit environ 2 Mb à chaque instant t) pourraient être également envoyées à afficher ou à sauvegarder sur un ordinateur hôte.

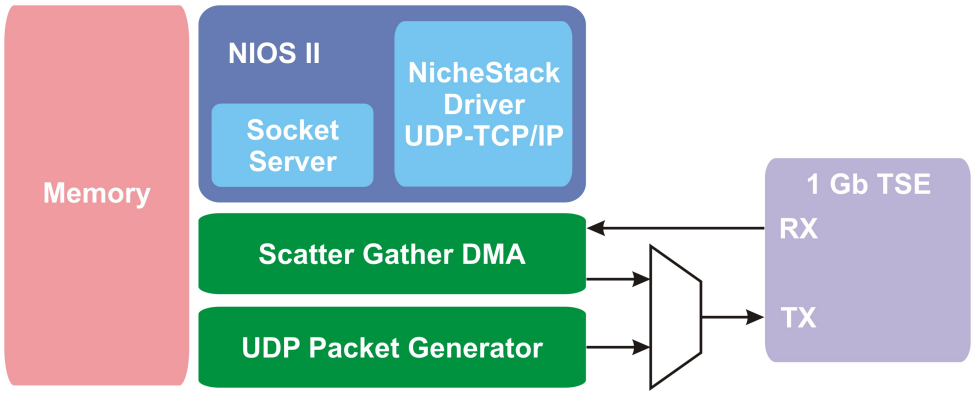


FIGURE 5.22 – L'architecture du module de communication pour le TSE IP.

Le lien Ethernet Gb a été choisi pour envoyer des données à partir de notre unité de traitement à l'ordinateur hôte. Nos premiers essais avec le pilote TCP Nichestack a permis d'atteindre 5 Mb/s de données utilisateur pour une communication fiable. Pour les images, le protocole UDP est suffisant. A cette vitesse on peut donc afficher environ 2 images/s. sur l'écran ded l'ordinateur hôte. Comme indiqué dans les notes d'application de chez Altera [26], nous avons désactivé le checksum et renforcé le processeur Nios II ; on devrait atteindre 36 Mb/s ce qui est encore loin de profiter à plein de la connexion 1 Gb.

Vu ces limitations, nous avons développé notre propre IP *UDP Packet Generator* qui pourrait être utilisé seulement pour la transmission ; après la configuration des registres pour la localisation et la taille des données à transférer, cet IP fonctionne de manière autonome après une unique impulsion de lancement, il peut insérer les en-têtes nécessaires et générer le début et la fin des paquets nécessaires pour le protocole Avalon. Cet IP a permis d'atteindre le débit maximal possible aux alentours de 963 Mb/s de données utilisateur. En raison de certains problèmes de notre application sur PC pour gérer de tels niveaux, nous avons ajouté un paramètre facultatif pour rajouter un délai entre paquets et ajuster ainsi le taux nécessaire.

Architecture pour l'acquisition des images et la fusion

Les résultats décrits ci-dessous, ont été obtenus en collaboration étroite avec P.Lacroix, assistant-ingénieur qui s'occupe de l'intégration matérielle sur le robot (câblage, interfaçage des caméras Aptina LVDS...), et avec W.Filali et D.Botero, doctorants dont les sujets de thèse concernent aussi des systèmes intégrés de vision, l'un pour la vidéo-surveillance, l'autre pour la stéréovision temps réel.

Les acquisitions sur toutes les caméras sont contrôlées par un seul signal de déclenchement afin de les synchroniser. Le Write's module est un élément clé que nous avons développé pour répondre à certains problèmes. L'écriture sur la mémoire DDR SDRAM atteint la plus haute performance avec un accès séquentiel ; l'accès aléatoire peut diminuer la bande passante jusqu'à huit fois. Il est donc capital

de stocker un peu de données dans une FIFO dans chacun des Write's modules ; ce module reproduit un comportement de déclenchement de Schmitt entre ses limites de presque plein et presque vide. Tout Write's module accède au bus pour délivrer une quantité définie de mots, et chaque module, une fois qu'il a accédé au bus, verrouille l'arbitrage de bas niveau pour garantir la continuité de son écriture. Par ce comportement, l'arbitrage de haut niveau réparti entre les Write's moduleset il n'est pas nécessaire de centraliser sa gestion.

Dans la figure 5.23, chaque caméra délivre un flux de données spécifique codée sur la mosaïque Bayer. Ce flux est converti par Capture's module en un flux RGB Avalon-ST. Ensuite, il peut être traité par notre architecture de classification décrite dans les sections précédentes.

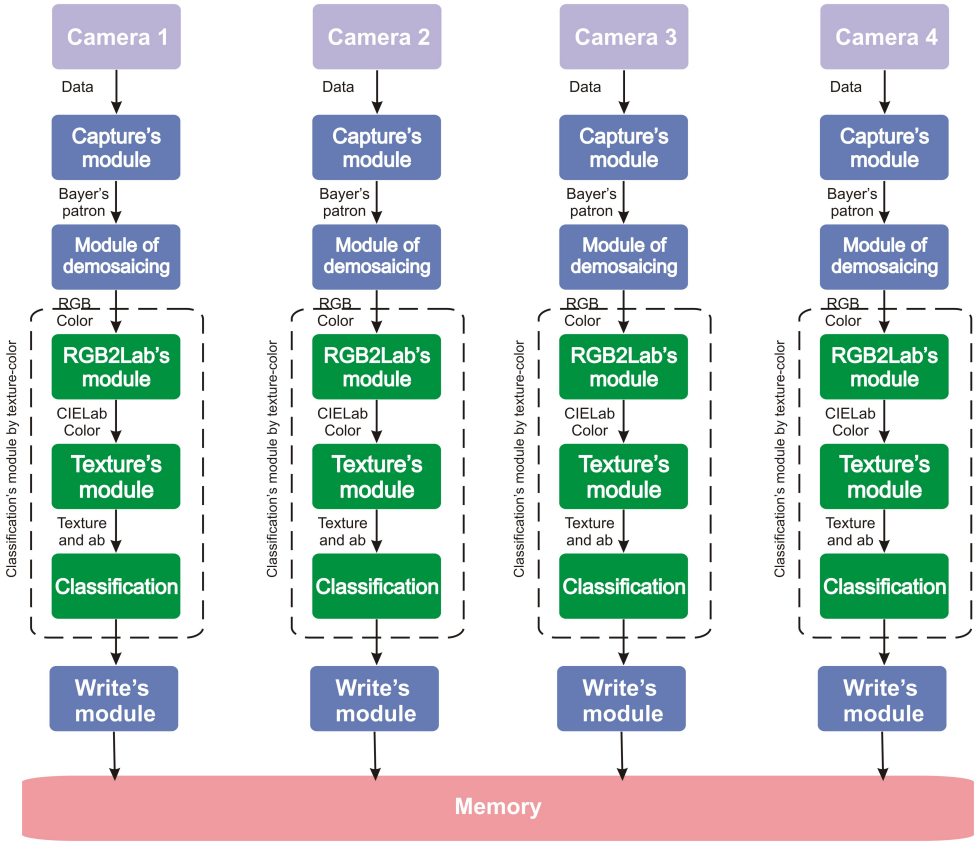


FIGURE 5.23 – L'architecture pour le traitement parallèle de quatre images .

Il est important de trouver le bon compromis pour le temps de commutation entre les différents Write's modules. Ce compromis est défini par la distance entre le presque plein et le presque vide de son comportement de déclenchement de Schmitt (distance de déclenchement), et la taille totale de la mémoire FIFO. Avec 4 Write's modules, pour le pire des cas lorsque la somme de leur bande passante est égale à la largeur de la bande mémoire partagée, alors la taille de la FIFO ne doit pas être inférieure à 4 fois la distance de déclenchement.

La mémoire partagée présentée en bas de la figure 5.23 va contenir les images de détection produites par les classifications appliquées sur les 4 images en parallèle. Elle est ensuite utilisée pour un traitement ultérieur, le Fetch's module décrit dans la figure 5.24. Ce module est en charge de fusionner directement ces images de détection dans la grille d'occupation ; c'est une simplification vis-à-vis de l'architecture globale présentée au chapitre 3 en figure 3.1. Toutes les images de détection sont directement fusionnées dans la grille globale, sans construire les grilles locales propres à chaque caméra.

Le Fetch's module est une autre composante conçue pour lire les pixels de la mémoire, non de façon séquentielle, mais en suivant les indications d'une table d'adresses, table d'indirections injectée dans ce module à l'aide du protocole Avalon-ST. Pour notre application, il apparaît deux cas selon que ces tables d'adresses sont statiques ou dynamiques. Si la grille d'occupation est robot-centrée, la table est statique : elle peut être tout simplement initialisée hors-ligne à partir des données de calibrage ; cette table combine les transformations homographiques entre plan du sol et plans image, avec les corrections de distorsions, comme cela a été décrit en section 3.4. Elle indique donc la correspondance entre

- les coordonnées métriques (x, y) d'une cellule dans la grille, coordonnées indiquées dans le repère robot,
- et les coordonnées pixel (u_i, v_i) dans l'image de détection construite à partir de la caméra i.

Pour construire la grille d'occupation, la grille est parcourue ; pour chaque (x, y), le Read's module lit la table d'adresses et récupère le triplet i et (u_i, v_i) qui indique dans quel image et dans quel pixel de cette image se trouve le rayon optique qui intersecte le sol en cette cellule. Cette adresse mémoire est injectée dans le Fetch's module qui peut accèder l'ensemble des adresses (u_i, v_i) des images dans la mémoire partagée. Notons que dans l'architecture actuelle, un seul rayon optique est associé à chaque cellule du sol : dans les zones de recouvrement des champs de vue, le rayon qui a l'angle d'incidence le plus grand est choisi.

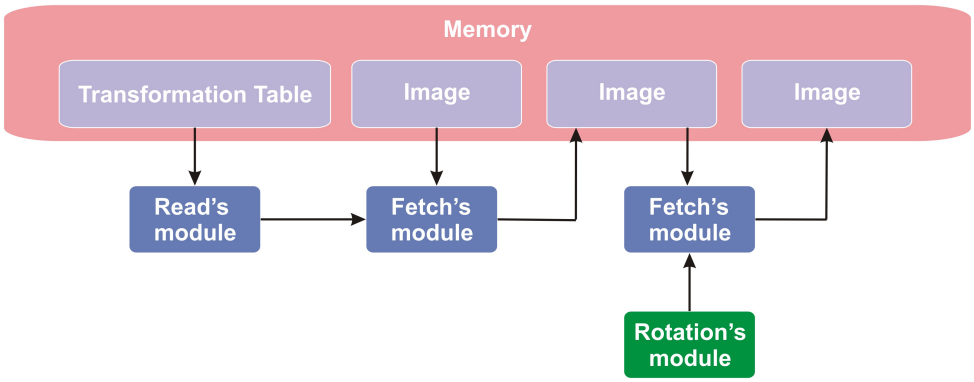


FIGURE 5.24 – L'architecture pour la rotation et la fusion des images.

Dans le cas où la grille d'occupation est construite dans le repère global, alors, la transformation elle-même n'est pas statique, car l'homographie entre les plans image et le plan du sol exprimé dans le repère global, dépend de la localisation du robot dans l'environnement, donc en particulier de l'orientation du robot. La table d'adresses injecté au Fetch's module peut être générée dynamiquement avec un composant spécifique appelé Rotation's module figurant à droite en figure 5.24. Pour la transformation de rotation, pour éviter le gaspillage d'utiliser la logique pour calculer des paramètres précis pour la matrice de rotation, la moitié du travail a été déchargée sur le processeur Nios II. Le processeur va calculer, sur chaque image, les paramètres de la matrice de rotation et les transférer au Rotation's module, qui multiplie simplement les coordonnées de chaque pixel par les éléments de la matrice en utilisant des multiplieurs embarqués, avant de convertir ces coordonnées en adresse de mémoire pour le Fetch's module.

Actuellement, notre système fonctionne sur le robot Scout avec 4 caméras présenté en figure 3.25. Le robot est déplacé par un opérateur à l'aide du joystick ; quatre images de détection sont construites en parallèle par l'architecture pour la détection d'obstacles fondée sur l'apparence. Une grille est construite à 30Hz, grille qui est soit robot-centrée, soit exprimée dans le repère du monde. Signalons deux problèmes que nous essaierons de résoudre rapidement :

– d'abord nous n'avons pas pu pour diverses raisons, sortir des figures (copies d'écran, vidéo ou autres) montrant la grille construite en temps réel. Ces résultats seront rajoutés dans la version finale.
– ensuite, nous avons eu une difficulté pour fusionner la grille construite à l'instant t avec celle construite à l'instant $t-1$. De ce fait, pour l'heure, la fusion temporelle des résultats de la détection n'est pas faite.

L'objectif pour la version finale est de faire fonctionner ce système avec 8 micro-caméras Aptina connectées au kit par des liaisons LVDS ; ce travail est en cours.

5.4 Conclusions

Ce chapitre a décrit comment une méthodologie complexe de classification a été mise en œuvre sur une architecture à base de FPGA. Il a été prouvé que les mêmes résultats sont obtenus avec une implémentation sur logiciel et avec la mise en œuvre sur matériel. Cet effort de concevoir, développer et évaluer cette architecture est justifiée pour des raisons différentes, principalement pour parvenir (1) à des performances temps réel pour traiter en parallèle 4 images de 640×480 pixels aujourd'hui, 8 à terme, (2) la portabilité sur un robot mobile : le kit Stratix3 et la ceinture (partielle encore) de caméras sont sur un robot Scout et (3) la consommation d'énergie. Actuellement, l'architecture traite 30 images par seconde, cette performance est liée à l'application (c'était la contrainte fixée à l'origine) et aux caractéristiques des cameras choisies (c'est la limite sur les TERASICs). En utilisant une implémentation logiciel dans seulement un PC, pour la même méthode, il n'est pas possible maintenant, de classifier 30 images de 640×480 pixels par seconde, même avec un code optimisé.

Nous pourrions encore améliorer cette performance temps réel en « bridant » la classification par réduction du nombre des attributs au risque de dégrader les résultats de détection (plus de faux positifs ou faux négatifs). Dans l'architecture pour l'analyse des attributs de texture, en utilisant la méthode proposée ASDH, le temps de calcul, les besoins en mémoire et le temps de latence sont considérablement réduits, principalement parce que le calcul des histogrammes est évité. En plus, la mémoire requise est plus petite que les algorithmes basés sur les techniques habituelles de HSD ou GLCM. Tous ces avantages permettent de mener à bien le traitement d'images en temps réel. Les attributs de texture (moyenne, contraste, homogénéité) sont calculés à 100 MHz. Si seulement ces trois attributs de texture sont utilisés par l'architecture, l'architecture peut atteindre une performance de 300 fps et 520 Gbps. Contrairement, les attributs de texture variance et corrélation, qui nécessité de calculer la pseudo-variance, sont calculés à 10 MHz, ce qui limite la performance de l'architecture pour l'analyses de la texture. Ainsi, le module pseudo-variance doit être amélioré pour accroître la performance.

Dans la classification, le processus d'apprentissage joue un rôle crucial pour trouver le classifieur avec le meilleur compromis entre la performance dans la classification et les ressources consommées. Afin d'accélérer ce processus, une méthodologie et les outils nécessaires ont été développés, ce qui permet d'optimiser le processus et de tester d'une façon plus rapide les différents modules de classification. Nous pouvons aussi adapter l'architecture de détection des obstacles à différents environnements et ou même pour d'autres applications. Par exemple, il est possible de modifier le champ d'application de notre architecture, par exemple, dans des applications médicales pour la détections des maladies par imagerie ou dans des applications militaires pour la détection de cibles dans un environnement naturel.

TABLE 5.5 – Table comparative de la performance pour différentes architectures sur FPGA.

	Our Design	Ibarra-Pico [85]	Tahir [132]	Maroulis [102]	Siéler [127]
Algorithme d'analyses de la texture	ASDHs	GLCH	GLCM	GLCM	GLCM
Taille de l'image $[U \times V]$ (pixels)	640×480	512×512	512×512	352×288	512×512
Nombre de bandes de couleur $[B]$	1	1	16	4	1
Taille de la fenêtre de traitement $[K \times L]$ (pixels)	17×15	32×32	128×128	16×16	128×128
Nombre de niveaux dans chaque bande $[N_g]$	256	32	32	64	32
Nombre des attributs de texture $[f]$	6	1	7	4	6
Nombre de déplacements $[d]$	4	4	16	4	16
Performance $[p]$ (fps)	30	$25 - 30$	$1.58 + 13.88$	133	27
Performance de traitement [pp] (bps)	52 Gbps	19 Gbps	442 Mbps	617 Mbps	405 Mbps
Latence (μs)	449.8	–	–	–	–
Technologie	FPGA	FPGA+DSP	FPGA	FPGA	FPGA
Logiques Eléments (Altera)	$73,021$	–	–	–	–
Nombre des multiplicateurs embarquées	480	–	–	–	–
Slices (Xilinx)	–	–	$11,328 + 9,216$	$16,158$	$5,740$
Taille de la mémoire interne	720 Kb	–	360 Kb	160 Kb	900 Kb
Taille de la mémoire externe	–	1 Mb +1 Mb	8 Mb	8 Mb	–

Conclusion générale et perspectives

Bilan des travaux réalisés

Les travaux présentés dans ce document s'intéressent à la conception d'un capteur visuel multi-caméras dédié à la détection d'obstacles ; notre objectif a été d'intégrer ce capteur sur un robot de service exécutant des mouvements en milieu intérieur humain. La diffusion massive de caméras dans les téléphones mobiles et les ordinateurs portables a permis de réduire considérablement le coût des micro-caméras ; les roboticiens doivent tirer profit de cette situation pour concevoir et développer des systèmes visuels passifs à bas prix, donc des systèmes exploitables sans danger en milieu humain, et qui donne des informations très riches. Cependant, la complexité algorithmique et la puissance de calcul requise sont très élevées afin de profiter de la richesse de l'information contenue dans les images, et cela est encore accentué pour les systèmes multi-caméras.

Ainsi, deux étapes majeures dans la conception du système ont été présentées dans ce document : d'une part le développement et la validation de l'algorithme de traitement pour la détection d'obstacles et d'autre part, la conception et l'implémentation de l'architecture sur un système dédié. Mais nous avons aussi évalué plusieurs outils pour la phase de transition de l'algorithme à l'architecture ; il est essentiel de choisir une méthodologie pour cette transition qui permette d'atteindre le meilleur compromis entre la performance et les ressources nécessaires, et de tirer le meilleur bénéfice à la fois de l'algorithme et de l'architecture.

Algorithmes pour la détection d'obstacles

Deux méthodes complémentaires de détection d'obstacles ont été présentées.

(1) La première méthode repose sur l'apparence des objets ; cette méthode permet d'utiliser une connaissance a priori de l'environnement pour classer chaque région de l'image comme image du SOL ou comme image d'un OBSTACLE en utilisant les attributs de couleur et texture. Cette méthode se compose de trois principaux algorithmes. Le premier algorithme permet de calculer les attributs de couleur ; nous avons choisi l'espace CIE-Lab, parce que nous cherchons un espace de couleur qui a la meilleure immunité possible aux changements d'éclairage.

Le deuxième algorithme permet de calculer les attributs de texture. Nous avons choisi les histogrammes de sommes et de différences, car cette technique était déjà connue dans notre équipe et avait donné des résultats acceptables lors de travaux antérieurs. Cet algorithme *classique* a le désavantage comme la plupart des algorithmes d'analyse de la texture, d'être très gourmand en puissance de calcul et ressources mémoire. Dans le cas de notre technique, ce sont les histogrammes eux-mêmes qui consomment le plus de ressources mémoire ; en plus leur calcul nécessite un grand nombre de multiplicateurs. C'est pourquoi nous avons décidé d'adapter la technique pour éviter le calcul des histogrammes ; les attributs de texture sont calculés d'une manière plus directe. Nous avons donc proposé une adéquation algorithmique pour optimiser son fonctionnement sur des systèmes parallèles.

Enfin, le troisième algorithme fait la classification basée sur la technique AdaBoost, utilisée pour classer les pixels selon leurs attributs de couleur et texture. Nous avons choisi cette technique pour sa*plus* facile mise en œuvre sur un système dédié. C'est important de noter que tous ces algorithmes ont été sélectionnés pour leur future mise en œuvre dans une architecture sur FPGA. Une analyse approfondie de la performance de la méthode de classification a permis d'observer les effets de chaque paramètre sur la performance globale ; cette analyse s'est révélée indispensable pour la sélection de ces paramètres sur l'architecture dédié finale.

(2) La deuxième méthode de détection d'obstacles utilise le mouvement du robot afin de détecter les obstacles. Contrairement à la première méthode qui peut être étendue à l'identification des objets à partir des attributs photométriques, cette deuxième méthode ne peut que détecter les obstacles. C'est une approche géométrique qui exige un étalonnage précis des caméras et une information fiable sur les déplacements du robot ; elle permet de détecter la hauteur des objets et donc de qualifier le danger qu'ils représentent pour la navigation du robot. Enfin, nous avons présenté la fusion des deux méthodes et la création d'une grille d'occupation utilisée à terme pour le suivi, pour l'estimation de la vitesse des obstacles et pour la fusion temporelle.

Implémentation matérielle

Le chapitre décrit le positionnement, les avantages et les inconvénients de la technologie reconfigurable fondée sur des FPGAs. Puis, nous avons présenté la méthode *classique* utilisée pour concevoir et implémenter des applications en utilisant cette technologie ; l'un des principaux inconvénients de cette méthode est le temps nécessaire pour donner de résultats *acceptables*, au niveau de la performance ou de la surface. Pour cette raison, nous avons évalué une autre approche, la synthèse de haut niveau qui permet d'accélérer le processus de conception et donc de réduire sa durée. Cependant, les outils de synthèse sont relativement récents, ils ne s'appliquent pas dans tout les cas, surtout si les contraintes sur la performance ou sur la surface sont très fortes. Néanmoins, la synthèse de haut niveau est idéale pour une exploration architecturale et pour fournir une première solution qui peut ensuite être optimisée. À la fin de ce chapitre nous avons présenté une analyse comparative entre les deux méthodes de conception pour l'algorithme de stéréovision ; d'abord nous avons appliqué successivement ces méthodes pour concevoir et implémenter deux architectures dédiées à la stéréovision, puis nous avons comparé ces deux architectures au niveau de leur performance, leur surface et le temps de conception. Cette analyse a été approfondie en comparant ces deux architectures avec quelques architectures trouvées dans la littérature.

Il existe deux problèmes essentiels pour la mise en œuvre d'algorithmes de vision sur des architectures embarqués. D'abord quel algorithme est le plus adapté pour être intégré dans une architecture dédiée ? Normalement, les algorithmes de vision sont conçus et réalisés de manière séquentielle de sorte que leur mise en œuvre sur une architecture parallèle devient compliquée. Ensuite comment tirer le meilleur parti de l'architecture en fonction de l'algorithme, pour optimiser l'énergie, les ressources et la performance ? Cette démarche a été illustrée par l'adéquation algorithmique proposée pour l'analyse de texture, adéquation qui permet d'une part de réduire les ressources et d'autre part d'optimiser la performance.

L'architecture pour la détection d'obstacles est composée de trois grandes unités de calcul. La première est la transformation de couleur qui est réalisée par deux modules, un premier pour la transformation de l'espace de couleur RGB à l'espace primaire XYZ, un second pour la transformation non linéaire de l'espace de couleur XYZ à l'espace CIE-Lab. Les composantes chromatiques a et b sont utilisée comme des attributs de couleur lors de la classification. La luminance L est utilisé pour calculer les attributs de texture.

La deuxième unité de calcul traite de l'analyse de texture dans sept modules. Un premier module

calcule la somme et la différence du pixel courant avec un pixel de son voisinage, défini par un déplacement donné. Ce module peut calculer en parallèle jusqu'à seize déplacements prédéfinis. Ensuite, les images de sommes et de différences sont utilisées pour calculer cinq attributs de texture : la moyenne, le contraste, l'homogénéité, la variance et la corrélation. La moyenne est calculée directement à partir l'image de somme, de même le contraste et l'homogénéité s'obtiennent directement à partir de l'image de différence. Dans ces trois modules nous pouvons réutiliser les valeurs précédentes des attributs afin d'optimiser le calcul. Pour les attributs de variance et de corrélation, nous avons besoin de calculer la pseudo-variance, qui change pour chaque fenêtre de traitement ; nous avons donc développé un module qui calcule la pseudo-variance pour chaque fenêtre de traitement.

Enfin, une unité de calcul traite de la classification avec plusieurs modules. Un module construit la base d'apprentissage hors ligne, de façon automatique afin de réduire au maximum les possibles erreurs humaines. La base est créée directement depuis les images acquises par le FPGA ou indirectement depuis une base d'images acquises par ailleurs. A partir de cette base, l'outil d'apprentissage génère le classifieur AdaBoost, sous la forme d'un module de haut niveau et de T modules de bas niveau. Ces modules de bas niveau sont configurés pour implémenter des arbres de décision ; chacun génére une hypothèse faible. Toutes ces hypothèses sont combinées par le module de haut niveau.

La dernière partie de ce manuscrit présente le travail d'intégration du système multi-caméras de détection d'obstacles, avec seulement quatre caméras. Nous avons montré l'architecture globale ; les quatre flux vidéo sont traités de façon parallèle par quatre architectures de détection d'obstacles basée sur l'apparence. Puis ces quatre images classées sont fusionnées pour obtenir une grille d'occupation qui peut être utilisée lors de la navigation de notre robot.

Perspectives

Enfin, évoquons des travaux que nous pourrions mener dans le futur sur la thématique présentée dans cette thèse. L'état actuel du prototype permet le traitement de quatre flux vidéo ; cette limitation vient du nombre maximal d'entrées dans le kit de développement. C'est pourquoi nous travaillons dans l'intégration de caméras avec une connexion LVDS exploitant le protocole I2C. Cela augmentera le nombre de ports disponibles ; comme la distance caméra-kit peut être en ce cas de plusieurs m., nous pourrons mieux positionner les caméras autour du robot, et aussi, considérer d'autres applications telles que la vidéo-surveillance (N caméras disposées dans une pièce par exemple).
Citons quelques perspectives en ce qui concerne l'architecture de détection d'obstacles basée sur l'apparence :

- Diverses optimisations, notamment sur le module de pseudo-variance, et sur le nombre de multiplieurs requis pour l'analyse de texture. L'architecture proposée utilise deux multiplieurs pour chaque pixel, parce que chaque pixel est codé sur 10 bits. Nous proposons d'utiliser un seul multiplieur au lieu de deux, en réduisant les bits significatifs des pixels de 10 à 9.
- L'architecture d'analyse de texture actuelle calcule seulement six des huit attributs de texture proposés par Unser. Il conviendrait de générer aussi des modules pour calculer les deux autres attributs : les clusters d'ombre et la proéminence.
- L'image des pixels classés est binarisée après application d'un seuil sur la sortie du classifieur ; cette image pourrait être améliorée soit par des opérations morphologiques pour réduire le nombre de pixels isolés mal classés, soit par un processus de filtrage comme le filtre médian.

Pour une question de temps, nous n'avons pas finalisé certains développements, ou nous les avons partiellement évalués.

- La grille d'occupation est actuellement calculée pour le temps courant, c'est à dire, nous calculons

la grille pour la détection courante sans fusionner avec les grilles précédentes. La fusion des grilles devrait améliorer la performance du système ; ce module aura besoin de l'information odométrique et d'un étalonnage des caméras.
- La deuxième méthode de détection basée sur l'analyse spatio-temporelle, a été testée uniquement en logiciel. L'implémentation de cette technique permettra de filtrer les nombreuses fausses détections que nous avons constatées avec uniquement la méthode basée sur l'apparence.

Au niveau de la méthodologie de conception,
- Il faudrait poursuivre des travaux sur l'évaluation des outils de synthèse de haut niveau en générant des architectures à partir des mêmes algorithmes, et en comparant ces architectures avec celle décrite dans ce manuscrit.
- Pour cette étude, nous avons choisi d'implémenter nos algorithmes sur FPGA Altera Stratix3. La méthodologie peut-elle aider au choix de la cible, technologie FPGA différente (Xilinx), ou autres technologies embarquées comme DSP ou GPU, en tenant compte non seulement de la performance mais aussi de la consommation d'énergie ?

Enfin, il conviendrait d'évaluer notre ceinture de micro-caméras par de nombreuses expérimentations sur des robots mobiles évoluant en milieu humain. Vis-à-vis des classiques ceintures de capteurs ultrasons, la vision offre une richesse qui n'est pour l'instant pas exploitée : par exemple les images devraient aussi permettre d'identifier la nature des obstacles.

Bibliographie

[1] IEEE standard VHDL language reference manual. 10.1109/IEEESTD.1994.121433, 1994.

[2] *Simulink HDL CoderTM 2 User's Guide*. The MathWorks, Inc., 2010.

[3] Roland Airiau, Jean-Michel Bergé, Vincent Olive, and Jacques Rouillard. *VHDL, langage, modélisation, synthèse, 2ème édition*. Presses Polytechniques et Universitaires Romandes (PPUR), 2 ème éd. rev. et augmentée edition, avril 1998.

[4] David Alleysson, Sabine Süsstrunk, and Jeanny Hérault. Color Demosaicing by Estimating Luminance and Opponent Chromatic Signals in the Fourier Domain. In *Proc. IS&T/SID 10th Color Imaging Conference*, volume 10, pages 331–336, 2002.

[5] Dora Almanza-Ojeda, Victor Ayala-Ramirez, Raul Sanchez-Yanez, and Gabriel Avina-Cervantes. *Performance Evaluation of a Segmentation Algorithm for Synthetic Texture Images*, volume 3789/2005, pages 379–385. Springer Berlin / Heidelberg, 2005.

[6] R. Andraka and A. Berkun. FPGAs make a radar signal processor on a chip a reality. In *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, volume 1, pages 559–563 vol.1, 1999.

[7] Miguel Arias-Estrada and Juan M. Xicotencatl. Multiple stereo matching using an extended architecture. In G. Brebner and R. Woods, editors, *FPL '01 : Proceeding of the 11th International Conference on Field-Programmable Logic and Applications*, pages 203–212, London, UK, 2001. Springer-Verlag.

[8] V. Athitsos and S. Sclaroff. Boosting nearest neighbor classifiers for multiclass recognition. In *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, page 45, 2005.

[9] J.G. Avina-Cervantes. *Navigation visuelle d'un robot mobile dans un environnement d'extérieur semi-structuré*. PhD thesis, Institut National Polytechnique de Toulouse, Feb. 2005.

[10] J.G. Avina-Cervantes, M. Estudillo-Ayala, S. Ledesma-Orozco, and M.A. Ibarra-Manzano. Boosting for image interpretation by using natural features. In *Artificial Intelligence, 2008. MICAI '08. Seventh Mexican International Conference on*, pages 117 –122, oct. 2008.

[11] Mario R. Barbacci and Daniel P. Siewiorek. Automated exploration of the design space for register transfer (RT) systems. In *Proceedings of the 1st annual symposium on Computer architecture*, pages 101–106. ACM, 1973.

[12] Massimo Bertozzi, Alberto Broggi, Alessandra Fascioli, and Ra Fascioli. Stereo inverse perspective mapping : Theory and applications. *IMAGE AND VISION COMPUTING JOURNAL*, 8 :585—590, 1998.

[13] Vaughn Betz and Jonathan Rose. FPGA routing architecture : segmentation and buffering to optimize speed and density. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 59–68, Monterey, California, United States, 1999. ACM.

[14] J.L. Boizard, M. Devy, P. Fillatreau, J.Y. Fourniols, P. Lacroix, N. Nasreddine, F.X. Bernard, and T. Sentenac. Real-time stereovision by an integrated sensor. In *6th IFAC Symp. on Intelligent Autonomous Vehicles (IAV 2007)*, pages 1–7. IFAC, 2007.

[15] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, Pittsburgh, Pennsylvania, United States, 1992. ACM.

[16] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, 1 edition, Janvier 1984.

[17] A. Broggi, M. Bertozzi, A. Fascioli, and S. Nichele. Stereo vision-based vehicule detection. *Intelligent Vehicules Symposium*, pages 34–44, Octobre 2000.

[18] Emmanuel Casseau, Bertrand Le-Gal, Pierre Bomel, Chistophe Jego, Sylvain Huet, and Eric Martin. C-based rapid prototyping for digital signal processing. In SuviSoft Oy Ltd., editor, *13th European Signal Processing Conference*, Antalya, Turkey, September 4-8 2005.

[19] M. Celenk. Analysis of color images of natural scenes. *Journal of Electronic Imaging*, 4(4) :382–396, October 1995.

[20] C. Chavet, C. Andriamisaina, P. Coussy, E. Casseau, E. Juin, P. Urard, and E. Martin. A design flow dedicated to multi-mode architectures for DSP applications. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 604–611, 2007.

[21] François Gauthier. Les outils de synthèse de haut niveau. *ElectroniqueS*, (201) :54–58, April 2009.

[22] Actel Corporation. Actel proasic fpga family data sheet. Technical report, Actel Corporation, 2000.

[23] Altera Corporation. Apex cpld family data sheet. Technical report, Altera Corporation, 2000.

[24] Altera Corporation. Stratix cpld family data sheet. Technical report, Altera Corporation, 2002.

[25] Altera Corporation. Cyclone ii device handbook, volume 1. Technical report, Altera Corporation, 2008.

[26] Altera Corporation. Accelerating nios ii networking applications, an-440-1.1. Technical report, Altera Corporation, March 2009.

[27] Altera Corporation. *Avalon Interface Specifications*. Altera Corporation, Août 2010.

[28] Altera Corporation. *Quartus II Handbook Version 10.0, Design and Synthesis*, volume 1. Altera Corporation, juillet 2010.

[29] Altera Corporation. *Quartus II Handbook Version 10.0, SOPC Builder*, volume 4. Altera Corporation, juillet 2010.

[30] Altera Corporation. Stratix iii device handbook, volume 1. Technical report, Altera Corporation, 2010.

[31] Xilinx Corporation. Xc4000 fpga family data sheet. Technical report, Xilinx Corporation, 1994.

[32] Xilinx Corporation. Virtex fpga family data sheet. Technical report, Xilinx Corporation, 1998.

[33] Xilinx Corporation. Virtex-ii fpga family data sheet. Technical report, Xilinx Corporation, 2001.

[34] Xilinx Corporation. Virtex-ii pro fpga family data sheet. Technical report, Xilinx Corporation, 2002.

[35] Philippe Coussy and Adam Morawiec. *High-Level Synthesis : from Algorithm to Digital Circuit*. Springer, 1 edition, October 2008.

[36] René de Jesús Romero Troncoso. *Electrónica digital y lógica programable*. Universidad de Guanajuato, 1er edition, 2007.

[37] M. Devy, M.-A. Ibarra-Manzano, J.L. Boizard, P. Lacroix, W. Filali, and J.Y. Fourniols. Integrated subsystem for obstacle detection from a belt of micro-cameras. In *Advanced Robotics, 2009. ICAR 2009. International Conference on*, pages 1–6, 2009.

[38] A.F. Dias, C. Lavarenne, M. Akil, and Y. Sorel. Optimized implementation of real-time image processing algorithms on field programmable gate arrays. In *Signal Processing Proceedings, 1998. ICSP '98. 1998 Fourth International Conference on*, volume 2, pages 1080–1083 vol.2, 1998.

[39] A. ELFES. A tesselated probabilistic representation for spatial robot perception and navigation. In California Inst. of Technology Propulsion Laboratory, editor, *JPL, California Inst. of Tech, Proceedings of the NASA Conference on Space Telerobotics,*, pages 341–350, Pasadena, California, Jan. 31-Feb. 2 1989.

[40] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6) :46–57, 1989.

[41] G. Estrin, B. Bussell, R. Turn, and J. Bibb. Parallel processing in a restructurable computer system. *Electronic Computers, IEEE Transactions on*, EC-12(6) :747–755, 1963.

[42] Gerald Estrin. Organization of computer systems : the fixed plus variable structure computer. In *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference*, pages 33–40, San Francisco, California, 1960. ACM.

[43] A. Ewald and V. Willhoeft. Laser scanners for obstacle detection in automotive applications. *Proceedings of IEEE Intelligent Vehicles Symposium*, Juin 2001.

[44] Stefan Florczyk. *Robot Vision : Video-based Indoor Exploration with Autonomous and Mobile Robots*. Wiley-VCH, april 2005.

[45] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application. *Journal of Computer and System Sciences*, 55(1) :119–139, August 1997.

[46] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression : a statistical view of boosting. Technical report, Dept. of Statistics, Stanford University, 1998.

[47] Jan Frigo, Maya Gokhale, and Dominique Lavenier. Evaluation of the streams-C C-to-FPGA compiler : an applications perspective. In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, pages 134–140, Monterey, California, United States, 2001. ACM.

[48] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin. *High-Level Synthesis : Introduction to Chip and System Design*. Kluwer Academic Publishers, Boston, MA, 1992.

[49] G. Gate, A. Breheret, and F. Nashashibi. Centralised fusion for fast people detection in dense environments. In *IEEE Int. Conf. on Robotics Automation (ICRA), Kobe, Japan*, 2009.

[50] Theo Gevers and Arnold W. M. Smeulders. Color-based object recognition. *Pattern Recognition*, 32(3) :453–464, March 1999.

[51] L. Gond, Q. C. Phanm, J. Begard, N. Allezard, and P. Sayd. Imagerie infrarrouge pour la surveillance de foules. *Reconnaissance des Formes et Intelligence Artificielle, RFIA*, pages 750–758, 2008.

[52] Paul Graham and Brent Nelson. FPGA-based sonar processing. In *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, pages 201–208, Monterey, California, United States, 1998. ACM.

[53] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers. Optimized generation of data-path from c codes for FPGAs. In *Design, Automation and Test in Europe, 2005. Proceedings*, volume 1, pages 112–117, 2005.

[54] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. SPARK : a high-level synthesis framework for applying parallelizing compiler transformations. In *VLSI Design, 2003. Proceedings. 16th International Conference on*, pages 461–466, 2003.

[55] Jiawei Han and Micheline Kamber. *Data Mining : Concepts and Techniques*. Morgan Kaufmann, 1st edition, September 2000.

[56] R.M. Haralick. Statistical and structural approaches to texture. *Proceedings of the IEEE*, 67(5) :786–804, may 1979.

[57] Robert M. Haralick and Linda G. Shapiro. *Computer and Robot Vision (Volume I)*. Addison Wesley Longman, 2002.

[58] Tim Harriss, Richard Walke, Bart Kienhuis, and Ed Deprettere. Compilation from matlab to process networks realized in FPGA. *Design Automation for Embedded Systems*, 7 :385–403, 2002. 10.1023/A :1020367508848.

[59] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN : 0521540518, second edition, 2004.

[60] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, corrected edition, July 2003.

[61] M. Hauta-Kasari, J. Parkkinen, T. Jaaskelainen, and R. Lenz. Generalized Co-Occurrence matrix for multispectral texture analysis. In *Pattern Recognition, International Conference on*, volume 2, pages 785–789, Los Alamitos, CA, USA, August 1996. IEEE Computer Society.

[62] Jeroen Hol. *Pose Estimation and Calibration Algorithms for Vision and Inertial Sensors*. PhD thesis, Department of Electrical Engineering, Linköping University, 2008.

[63] University of California at Riverside. http ://roccc.cs.ucr.edu/, March 2010.

[64] http ://sdi.acfr.usyd.edu.au/datasets/IRcameraCalibration/, Octobre 2010.

[65] GAUT High-Level Synthesis tool From C to RTL http ://www-labsticc.univ-ubs.fr/www gaut/, March 2009.

[66] Altera Corporation http ://www.altera.com/, March 2010.

[67] Altium http ://www.altium.com/, March 2010.

[68] Inc. http ://www.autoesl.com/, AutoESL Design Technologies, March 2010.

[69] Inc. http ://www.cadence.com/, Cadence Design Systems, March 2010.

[70] CebaTech Inc. http ://www.cebatech.com/, March 2010.

[71] Commission Internationale de l'Eclairage http ://www.cie.co.at/, July 2010.

[72] Impulse Accelerated Technologies http ://www.impulseaccelerated.com/, March 2010.

[73] http ://www.laas.fr/morse, Septembre 2010.

[74] Inc. http ://www.mathworks.com/, The MathWorks, March 2010.

[75] Mentor Graphics http ://www.mentor.com/, 2010.

[76] NEC Corporation http ://www.nec.com/, March 2010.

[77] National Instruments Corporation http ://www.ni.com/, March 2010.

[78] SynDex : System-Level CAS Software for Distributed Real-Time Embedded Systems http ://www.syndex.org/, March 2010.

[79] Inc. http ://www.synopsys.com/, Synopsys, March 2010.

[80] M.-A. Ibarra-Manzano, M. Devy, and J.-L. Boizard. Real-time classification based on color and texture attributes on an fpga-based architecture. In Adam Morawiec and Jinnie Hinderscheit, editors, *Proceedings of the 2010 Conference on Design and Architecture for Signal and Image Processing*, pages 53–60. Electronic Chips and Systems design Initiative, October 2010.

[81] M.-A. Ibarra-Manzano, M. Devy, J.L. Boizard, P. Lacroix, W. Filali, and J.Y. Fourniols. Obstacle avoidance by a multi-camera system. In MU. Mondragon Unibertsitateko Zerbitzu, editor, *Electronics, Control, Modelling, Measurement and Signals, 2009. ECMMS 2009. 9th. International Workshop on*, pages 81–87, Mondragon, España, July, 8-10 2009.

[82] M.A. Ibarra-Manzano, D.-L. Almanza-Ojeda, M. Devy, J.-L. Boizard, and J.-Y. Fourniols. Stereo vision algorithm implementation in fpga using census transform for effective resource optimization. In *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, pages 799 –805, aug. 2009.

[83] Mario A. Ibarra-Manzano, J. Gabriel Avina-Cervantes, Dora L. Almanza-Ojeda, and Jose Ruiz-Pinales. Detection of closure features in Gray-Level images by using support vector machines. *Telecommunications and Radio Engineering*, 64(11) :923–929, 2005.

[84] Mario Alberto Ibarra-Manzano, Michel Devy, Jean-Louis Boizard, Pierre Lacroix, and Jean-Yves Fourniols. An efficient reconfigurable architecture to implement dense stereo vision algorithm using high-level synthesis. In *2009 International Conference on Field Programmable Logic and Applications*, pages 444–447, Prague, Czech Republic, 2009.

[85] F. Ibarra Pico, S. Cuenca Asensi, and V. Corcoles. Accelerating statistical texture analysis with an fpga-dsp hybrid architecture. In *FCCM '01 : Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 289–290, Washington, DC, USA, 2001. IEEE Computer Society.

[86] B. Jida, R. Lherbier, J.-C. Noyer, and M. Wahl. Multiple target detection and tracking by interacting joint probabilistic data association filter and bayesian networks : Application to real data. In *Intelligent Transportation Systems, 2009. ITSC '09. 12th International IEEE Conference on*, pages 1–8, 2009.

[87] Linda Kaouane, Mohamed Akil, Yves Sorel, and Thierry Grandpierre. From algorithm graph specification to automatic synthesis of fpga circuit : A seamless flow of graphs transformations. In *Field-Programmable Logic and Applications*, volume 2778 of *Lecture Notes in Computer Science*, pages 934–943. Springer Berlin / Heidelberg, 2003.

[88] R. Katz, B. Douillard, J. Nieto, and E. Nebot. A self-supervised architecture for moving obstacles classification. pages 155 –160, sep. 2008.

[89] Vojislav Kecman. *Learning and Soft Computing : Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. The MIT Press, 1 edition, March 2001.

[90] A. Kelly, A. Stentz, O. Amidi, M. Bode, D. Bradley, A. Diaz-Calderon, M. Happold, H. Herman, R. Mandelbaum, T. Pilarski, P. Rander, S. Thayer, N. Vallidis, and R. Warner. Toward reliable off road autonomous vehicles operating in challenging environments. *International Journal of Robotics Research*, pages 449–483, Mai-Juin 2006.

[91] J. Klappstein, F. Stein, and U. Franke. Monocular motion detection using spatial constraints in a unified manner. In *Intelligent Vehicles Symposium, 2006 IEEE*, pages 261–267, 2006.

[92] Raphael Labayrade, Dominique Gruyer, Cyril Royere, Mathias Perrollaz, and Didier Aubert. Obstacle detection based on fusion between stereovision and 2d laser scanner. *Mobile Robots : Perception & Navigation*, pages 91–110, 2007.

[93] D. Langer. *An Integrated MMW Radar System for Outdoor Navigation*. PhD thesis, Carnegie Mellon University, The Robotics Institute, Pittsburg, Pensylvania, 1997.

[94] Dominique Lavenier. SAMBA : Systolic accelerator for molecular biological applications. Technical report, INRIA, March 1996.

[95] Dominique Lavenier, Frédéric Raimbault, and Patrice Frison. I/O and computation overlap on SIMD systolic arrays. Technical report, INRIA, November 1993.

[96] Vincent Lemonde. *Stéréovision Embarquée sur Véhicule : de l'Auto-Calibrage à la Détection d'Obstacles.* PhD thesis, INSA-LAAS-CNRS, 2005.

[97] D. T. Linzmeier, M. Skutek, M. Mekhaiel, and K. C. J. Dietmayer. A pedestrian detection system based on thermopile and radar sensor data fusion. *International Conference on Information Fusion*, pages 1272–1279, 2005.

[98] P. Longere, Xuemei Zhang, P.B. Delahunt, and D.H. Brainard. Perceptual assessment of demosaicing algorithm performance. *Proceedings of the IEEE*, 90(1) :123–132, janvier 2002.

[99] Santos Lopez-Estrada and Rene Cumplido. Decision tree based fpga-architecture for texture sea state classification. In *Reconfigurable Computing and FPGA's, 2006. ReConFig 2006. IEEE International Conference on*, pages 1–7, sept. 2006.

[100] Guangming Lu, Hartej Singh, Ming-hau Lee, Nader Bagherzadeh, Fadi Kurdahi, and Eliseu Filho. The morphosys parallel reconfigurable system. In Patrick Amestoy, Philippe Berger, Michel Daydé, Daniel Ruiz, Iain Duff, Valérie Frayssé, and Luc Giraud, editors, *Euro-Par'99 Parallel Processing*, volume 1685 of *Lecture Notes in Computer Science*, pages 727–734. Springer Berlin / Heidelberg, 1999.

[101] Hanspeter A. Mallot, H. H. Bülthoff, J. J. Little, and S. Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological Cybernetics*, 64(3) :177–185, 1991.

[102] Dimitris Maroulis, Dimitris K. Iakovidis, and Dimitris Bariamis. Fpga-based system for real-time video texture analysis. *J. Signal Process. Syst.*, 53(3) :419–433, 2008.

[103] Alan Marshall, Tony Stansfield, Igor Kostarnov, Jean Vuillemin, and Brad Hutchings. A reconfigurable arithmetic array for multimedia applications. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 135–143, Monterey, California, United States, 1999. ACM.

[104] E. Martin, O. Sentieys, H. Dubois, and J. L. Philippe. Gaut : An architectural synthesis tool for dedicated signal processors. In *Proceedings of the IEEE European Design Automation Conference*, pages 14–19. IEEE, September 20-24 1993.

[105] Grant Martin. Practical system-level design methodologies for processor-centric soc and embedded systems. In European Network of Excellence on High Performance, Embedded Architecture, and Compilation, editors, *International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems*, Barcelona, Spain, July, 12-18 2009. European Network of Excellence on High Performance and Embedded Architecture and Compilation, European Network of Excellence on High Performance and Embedded Architecture and Compilation.

[106] L. Matthies, T. Litwin, K. Owens, A. Rankin, K. Murphy, D. Coombs, J. Gilsinn, T. Hong, S. Legowik, M. Nashman, and B. Yoshimi. Performance evaluation of ugv obstacle detection with

ccd/flir stereo vision and ladar. *ISIC/CIRA/ISAS Joint Conference*, pages 658–670, Septembre 1998.

[107] L. Matthies, M. Mainmone, A. Johnson, Y. Cheng, R. Willson, C. Villalpando, S. Goldberg, A. Huertas, A. Stein, and A. Angelova. Computer vision on mars. *International Journal of computer Vision*, pages 67–92, Mars 2007.

[108] A. Mendes, L.C. Bento, and U. Nunes. Multi-target detection and tracking with a laser scanner. pages 796 – 801, jun. 2004.

[109] Thomas Mitchell. *Machine Learning*. McGraw Hill Higher Education, 1st edition, October 1997.

[110] Yosuke Miyajima and Tsutomu Maruyama. A real-time stereo vision system with fpga. In G. Brebner and R. Woods, editors, *FPL '03 : Proceeding of the 13th International Conference on Field-Programmable Logic and Applications*, pages 448–457, London, UK, 2003. Springer-Verlag.

[111] Chris Murphy, Daniel Lindquist, Ann Marie Rynning, Thomas Cecil, Sarah Leavitt, and Mark L. Chang. Low-cost stereo vision on an fpga. In *FCCM '07 : Proceeding of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 333–334, Washington, DC, USA, 2007. IEEE Computer Society.

[112] J. G. Avi na Cervantes. *Navigation visuelle d'un robot mobile dans un environnement d'extérieur semi-structuré.* PhD thesis, LAAS-CNRS, 2005.

[113] Walid Najjar. Opportunities and challenges of reconfigurable computing. In European Network of Excellence on High Performance, Embedded Architecture, and Compilation, editors, *International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems*, Barcelona, Spain, July, 12-18 2009. European Network of Excellence on High Performance and Embedded Architecture and Compilation, European Network of Excellence on High Performance and Embedded Architecture and Compilation.

[114] Abdelelah Naoulou, Jean-Louis Boizard, Jean Yves Fourniols, and Michel Devy. A 3d real-time vision sytem based on passive stereovision algorithms : Application to laparoscopic surgical manipulations. In *Proceedings of the 2nd Information and Communication Technologies, 2006 (ICTTA)*, volume 1, pages 1068–1073. IEEE, April 24-28 2006.

[115] Noboru Ohta and Alan Robertson. *Colorimetry : Fundamentals and Applications*. Wiley, Janvier 2006.

[116] Y. Ohta, Takeo Kanade, and T. Sakai. Color information for region segmentation. *Computer Graphics and Image Processing*, 13(1) :222 – 241, July 1980.

[117] Cameron Patterson. High performance DES encryption in virtex(tm) FPGAs using jbits(tm). In *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, page 113. IEEE Computer Society, 2000.

[118] Volnei A. Pedroni. *Circuit Design with VHDL.* The MIT Press, Agust 2004.

[119] D. Pomerleau. RALPH : rapidly adapting lateral position handler. In *Intelligent Vehicles '95 Symposium., Proceedings of the*, pages 506–511, 1995.

[120] Rajeev Ramanath and Wesley Snyder. Adaptive demosaicking. *JOURNAL OF ELECTRONIC IMAGING*, 12(4) :633—642, October 2003.

[121] R. Rinker, M. Carter, A. Patel, M. Chawathe, C. Ross, J. Hammes, W.A. Najjar, and W. Bohm. An automated process for compiling dataflow graphs into reconfigurable hardware. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 9(1) :130–139, 2001.

[122] Lior Rokach and Oded Maimon. *Data Mining with Decision Trees : Theroy and Applications*. World Scientific Publishing Co Pte Ltd, February 2008.

[123] T. Sakamoto, C. Nakanishi, and T. Hase. Software pixel interpolation for digital still cameras suitable for a 32-bit MCU. *IEEE Transactions on Consumer Electronics*, 44(4) :1342–1352, novembre 1998.

[124] Bernhard Schlkopf and Alexander J. Smola. *Learning with Kernels : Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 1st edition, December 2001.

[125] Herman H. Schmit, Srihari Cadambi, Matthew Moe, and Seth C. Goldstein. Pipeline reconfigurable fpgas. *Journal of VLSI Signal Processing Systems*, 24(2-3) :129–146, 2000.

[126] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.

[127] L. Siéler, C. Tanougast, and A. Bouridane. A scalable and embedded fpga architecture for efficient computation of grey level co-occurrence matrices and haralick textures features. *Microprocess. Microsyst.*, 34(1) :14–24, 2010.

[128] N. Simond and M. Parent. Obstacle detection from IPM and super-homography. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 4283–4288, 2007.

[129] Joan Sola. *Towards Visual Localization, Mapping and Moving Objects Tracking by a Mobile Robot : a Geometric Probabilistic Approach*. PhD thesis, LAAS-CNRS, 2007.

[130] STMicroelectronics. *IMG-724-E01 : Camera integration kit for 2 Megapixel CMOS single-chip camera module*, February 2007.

[131] S. Terho T. Peynot and S. Scheding. Sensor data integrity : Multi-sensor perception for unmanned ground vehicles. Technical report acfr-tr-2009-002, Australian Centre for Field Robotics (ACFR), The University of Sydney, 2009.

[132] M. A. Tahir, A. Bouridane, and F. Kurugollu. An fpga based coprocessor for glcm and haralick texture features and their application in prostate cancer classification. *Analog Integr. Circuits Signal Process.*, 43(2) :205–215, 2005.

[133] A. Talukder and L. Matthies. Real-time detection of moving objects from moving vehicles using dense stereo and optical flow. *International Conference on Intelligent Robots and Systems*, pages 3718–3725, Septiembre-Octubre 2004.

[134] Terasic. *TRDB_D5M : 5 Mega Pixel Digital Camera Development Kit*, March 2008.

[135] Donald E. Thomas and Philip R. Moorby. *The Verilog® Hardware Description Language*. Springer, 5th edition, 2002.

[136] S. Trimberger. Redefining the FPGA for the next generation. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, page 4, 2007.

[137] William Tsu, Kip Macy, Atul Joshi, Randy Huang, Norman Walker, Tony Tung, Omid Rowhani, Varghese George, John Wawrzynek, and André DeHon. HSRA : high-speed, hierarchical synchronous reconfigurable array. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, pages 125–134, Monterey, California, United States, 1999. ACM.

[138] M Unser. Sum and difference histograms for texture classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(1) :118–125, 1986.

[139] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, septembre 1998.

[140] Th. Veit, F. Cao, and P. Bouthemy. Space-time a contrario clustering for detecting coherent motion. In *IEEE Int. Conf. on Robotics and Automation, ICRA'07*, pages 33–39, Roma, Italy, April 2007.

[141] Chieh-Chih Wang, C. Thorpe, and A. Suppe. Ladar-based detection and tracking of moving objects from a ground vehicle at high speeds. pages 416 – 421, june 2003.

[142] T. Williamson. *A High-Performance Stereo Vision System for Obstacle Detection*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Septembre 1998.

[143] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *ECCV '94 : Proceedings of the Third European Conference on Computer Vision*, volume II, pages 151–158, Secaucus, NJ, USA, June 1994. Springer-Verlag New York, Inc.

Vision multi-caméras pour la détection d'obstacles sur un robot de service : des algorithmes à un système intégré.

Résumé : l'une des tâches les plus importantes en robotique mobile est la détection d'obstacles pendant les déplacements du robot. Pour résoudre cette tâche, de nombreuses approches ont été proposées; cependant les propositions applicables dans un milieu structuré, dynamique et fortement encombré du fait de la présence humaine, sont limitées. Dans ce cadre, nous présentons dans ces travaux un système visuel reprogrammable dédié à la détection d'obstacles.

Le système est composé de plusieurs micro-caméras disposées autour du robot mobile et d'un système reprogrammable. Le nombre de micro-caméras est grand (4 dans la version courante, 8 dans la version finale) et la performance en temps réel requis dans ce contexte, ne peut pas être satisfaite par un processeur standard. Cela rend obligatoire la conception et la mise en oeuvre d'une architecture dédiée pour le traitement des images. Le parallélisme fourni par les FPGAs permet de répondre aux contraintes de performance et de minimiser l'énergie et le coût unitaire du système.

L'objectif est de construire et mettre à jour une grille d'occupation robot-centrée lors de la navigation du robot. Cette opération doit être exécutée à 30Hz, afin de réduire la latence entre l'acquisition des images et la détection des obstacles.

La détection des zones du sol occupées est faite par l'algorithme de classification AdaBoost en utilisant un vecteur d'attributs. Les attributs utilisés sont la couleur et la texture. Pour la couleur, nous utilisons l'espace de couleur CIE-Lab, car cela permet d'avoir une plus grande immunité au changement de l'éclairage. Les attributs de texture sont obtenues par une méthode adaptée de la technique des histogrammes de sommes et différences. Cette adaptation réduit considérablement les ressources nécessaires pour calculer les attributs de texture, tout en fournissant un modèle riche de chacun des objets présents dans une scène acquise par une des micro-caméras. Chaque pixel dans l'image est classifié pour savoir s'il appartient ou pas au sol, en fonction de ces attributs couleur-texture. Une fois le pixel classé, il est projeté sur le plan du sol pour enrichir la grille d'occupation courante de l'environnement.

Plusieurs paramètres de notre approche ont été sélectionnés afin de développer un système avec le meilleur compromis entre les performances et les ressources consommées. Les graphiques de performances de la classification ainsi que les ressources consommées par les architectures implantées sont présentés. Les architectures ont été développées en VHDL avec les outils Altera; des comparaisons sont présentées avec une approche fondée sur des outils de synthèse haut-niveau (Gaut, labview...). Finalement ces architectures ont été portées et évaluées sur un kit Stratix3 connecté à 4 caméras et embarqué sur un robot mobile.

Mots-clés : Adéquation Architectures-Algorithmes, Vision multi-caméras, Classification couleur, texture, détection obstacles

Multi-cameras vision for obstacle detection for a service robot : from algorithms to an integrated system.

Abstract: one of the more important tasks to be executed on a mobile robot, concerns the detection of obstacles during the robot motions. Many methods have been proposed for this function: nevertheless their performances are limited when applied in a structured environment made highly dynamic and cluttered due to humans. This document presents a visual and flexible system for obstacle detection in such an environment.

The system is made of several micro-cameras fixed all around the robot body, and of a programmable electronic board. The camera number must be large enough (4 in the current version, 8 in the future one), so that real-time performances mandatory for such a function, cannot be reached from a standard multipurpose processor. It makes compulsory to design and to implement a hardware architecture devoted for image processing. The execution of parallel processes on FPGAs allows to reach real-time performances, while minimizing the required energy and the system cost.

The system objective consists in building and updating a robot-centered occupancy grid while the robot is navigating. This function must be executed at 30Hz, in order to minimize the latency between image acquisition and obstacle detection.

The detection of occupied ground areas is given by a classification algorithm, using an AdaBoost classifier on characteristic vectors. These vectors are built from color and texture attributes. For the color, the CIE-Lab space has been selected because it allows a better invariance according to the light variations. For the texture, an original method has been proposed adapting the Unser approach based on sum and difference histograms. This approach has been modified in order to reduce significantly the resources required to compute the texture attributes, while providing a fine model for every object detected on a scene acquired by each micro-camera. Each pixel in every image is classified as Ground or Obstacle, with respect to its color and texture attributes. Once a pixel is classified, it is projected on the ground plane in order to update the current occupancy grid built to represent the environment.

Many parameters for our approach have been selected in order to develop a system with the better trade-off between performances and consumed resources. Every proposed architecture is evaluated using curves between classification performances and required resources. These architectures have been developed in VHDL using the Altera tool boxes; this classical approach has been compared with a method based on tools providing high level synthesis (Gaut, labview...). Finally all architectures avec been implemented and evaluated on a Stratix3 development kit connected to four cameras, and embedded on a mobile robot.

Keywords: Adaptation between architectures and algorithms, Multi-cameras Vision, Color classification, texture, obstacle detection