Architecture générique pour le système de vision sur FPGA - Application à la détection de trait laser

Seher Colak

N°d'ordre NNT : 2018LYSES010

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
Laboratoire Hubert Curien

École Doctorale N°488
Sciences Ingénierie Santé

Spécialité de doctorat : Microélectronique

Soutenue à huis clos le 19/04/2018, par :
Seher Colak

Architecture générique pour système de vision sur FPGA

Application à la détection de trait laser

Devant le jury composé de :

M, El-Bay Bourennane	PR	Université de Dijon	Président
M, Jean François Nezan	PR	INSA de Rennes	Rapporteur
M, Dominique Houzet	PR	GIPSA lab de Grenoble	Rapporteur
Mme, Virginie Fresse	MC-HDR	Laboratoire Hubert Curien	Directrice de thèse
M, Olivier Alata	PR	Laboratoire Hubert Curien	Co-Directeur de thèse
M, Emmanuel Dumas	DR	Pattyn Bakery Division	Co-Directeur de thèse

Abstract

This thesis is part of an industrial research training agreement (CIFRE) between the Hubert Curien laboratory and the company Pattyn Bakery Division. The goal of this work is the development of an FPGA laser line detection system that is more efficient than the current system of the company. In the industry, vision system designers need to be able to easily create and modify their systems in order to adapt them to their customers' needs and technological developments. Thus developed operators must be generic to allow designers to change the vision system without necessarily having material skills. Designers must also be able to estimate what resources will be used by the operator in case of system changes : application parameters, sensor, family of FPGAs ...

In this manuscript, the main laser line detection algorithms and their properties have been studied. A laser line detection operator was chosen and developed. The implementation of this operator on an FPGA-camera from market has resulted in a first functional prototype. The time performance of this new system is four times that of the system currently used by the company. The new system is able to process up to 2500 frames per second. Finally, resource consumption models makes it possible to size an architecture from a set of predefined parameters quickly and without synthesizing. The parameter to which designers must pay the most attention is the level of parallelism of the data. This parameter makes it possible to exploit the parallelism capabilities of the FPGA by consuming more resources. However, the resources of the FPGA are limited and increasing the level of parallelism can induce the need to change the family of FPGAs. The system and the data provided will enable the company to adapt the vision system to the future needs of customers by guiding the choice of equipment.

Résumé

Cette thèse s'inscrit dans le cadre d'une convention industrielle de formation par la recherche (CIFRE) entre le laboratoire Hubert Curien et l'entreprise Pattyn Bakery Division. L'objectif de ces travaux est le développement d'un système de détection de trait laser sur FPGA (*Field Programmable Gate Array*) qui soit plus performant que système actuel de l'entreprise. Dans l'industrie, les concepteurs de systèmes de vision doivent pouvoir créer et modifier facilement leurs systèmes afin de pouvoir les adapter aux besoins de leurs clients et aux évolutions technologiques. Ainsi les opérateurs développés doivent être génériques afin de permettre aux concepteurs de modifier le système de vision sans nécessairement avoir de compétences matérielles. Les concepteurs doivent également pouvoir être en mesure d'estimer quelles seront les ressources utilisées par l'opérateur en cas modifications du système : paramètres de l'application, capteur, famille de FPGA...

Dans ce manuscrit, les principaux algorithmes de détection de trait laser ainsi que leurs propriétés ont été étudiés. Un opérateur de détection de trait laser a été choisi et développé. L'implantation de cet opérateur sur une caméra-FPGA du marché a permis d'obtenir un premier prototype fonctionnel. Les performances temporelles de ce nouveau système sont quatre fois supérieures à celles du système actuellement utilisé par l'entreprise. Le nouveau système est capable de traiter jusqu'à 2500 images par seconde. Enfin, les modèles de la consommation des ressources permettent de dimensionner une architecture à partir d'un ensemble de paramètres prédéfinis de manière rapide et sans faire de synthèses. Le paramètre auquel les concepteurs doivent prêter le plus d'attention est le niveau de parallélisme des données. Ce paramètre permet d'exploiter les capacités de parallélisme du FPGA en consommant plus de ressources. Cependant, les ressources du FPGA sont limitées et augmenter le niveau de parallélisme peut induire la nécessité de changer de FPGA. Le système et les données fournies permettront à l'entreprise d'adapter le système de vision selon les besoins futurs des clients en les guidant le choix du matériel.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1 : Introduction

1.1 Contexte général et motivations

Cette thèse a été réalisée dans le cadre d'une convention industrielle de formation par la recherche (CIFRE) avec l'entreprise Pattyn Bakery Division et le laboratoire Hubert Curien. Cette entreprise est spécialisée dans la conception et l'installation de lignes de conditionnement et de contrôle qualité pour la boulangerie et la viennoiserie industrielle. Ces lignes utilisent la vision industrielle et plus précisément la technique de la triangulation laser.

Dans ce contexte, les clients de Pattyn veulent des machines de plus en plus larges et de plus en plus rapides. Pour répondre à cette demande, il faut traiter des images de résolution plus élevée acquises à des vitesse plus importantes. De nos jours, les vitesses d'acquisition et résolutions des capteurs image permettent de répondre à cette demande. Le tableau 1.1 présente la résolution, la vitesse d'acquisition et le débit pixel de capteurs image dédiés aux applications de type machine vision qui permettent de répondre à cette demande. Cependant le système de vision actuellement mis en place dans l'entreprise a atteint ses limites techniques et ne permet pas de traiter un débit pixel aussi important.

Le système de vision actuel de l'entreprise est représenté sur la figure 1.1. Il est constitué d'une caméra qui acquiert des images de résolution 2048x128 pixels. Ces images sont transmises à une carte d'acquisition Silicon Software MicroEnable IV. La détection de trait laser est réalisée sur cette carte. Pour une image en entrée, la sortie est constituée de 2048 valeurs 16 bits qui correspondent à la position du trait laser sur l'image initiale. Cette ligne de résultats est transmise à un PC industriel via une connexion PCI Express.

La principale limite de ce système se situe dans la communication caméra-carte d'acquisition. Le tableau 1.2 présente des standards de transmission d'image entre caméras et cartes d'acquisition, ainsi que leurs débits et longueurs maximums. Dans le système de vision de l'entreprise, le débit maximum du capteur est 900 Mo/s quand le débit maximum permis par le câble Camera Link est de 850 Mo/s. Actuellement, le système de vision traite 600 images/s, alors que le capteur permet d'acquérir jusqu'à 2600 images/s. La limitation du système de vision en termes de débits ne permet pas à l'entreprise d'augmenter la résolution des images traitées, ni la vitesse d'acquisition d'images.

TABLE 1.1 – Capteurs image intéressants pour les applications machine vision

Référence	Résolution	Vitesse d'acquisition	Débit maximal
CMOSIS CMV2000	2048 x 1088	338 images/s	753 Mpixels/s
ON Semi VITA25K	5120 x 5120	53 images/s	1 389 Mpixels/s
ON Semi VITA12K	4096 x 3072	160 images/s	2 013 Mpixels/s
CMOSIS CMV12000	4096 x 3072	300 images/s	3 774 Mpixels/s

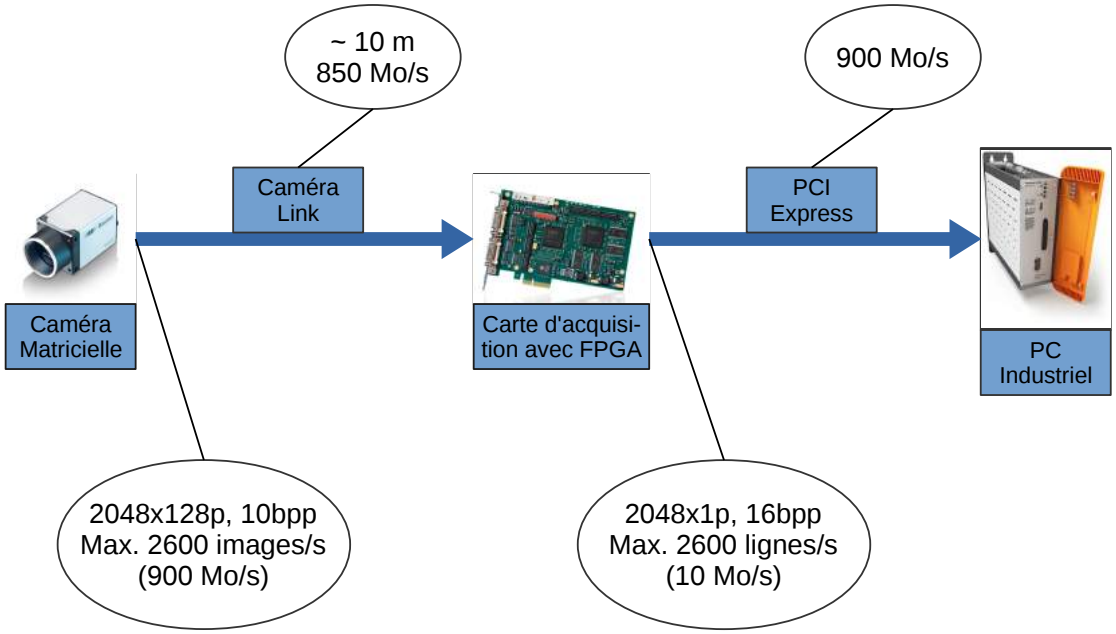


FIGURE 1.1 – Système de vision actuellement en place dans les machines Pattyn Bakery Division.

De plus, les câbles Camera Link ont quelques inconvénients. L'un de ces inconvénients est leur longueur maximum qui varie de 7 à 10 m. Cette contrainte de longueur de câble pose quelques problèmes lors du design des machines de l'entreprise. Le second inconvénient de ce câble est sa fragilité. Les câbles Camera Link les plus flexibles de nos jours ont un rayon de courbure de 5 cm. Mais la majorité des câbles disponibles ont un rayon de courbure de l'ordre de 10 cm. Ainsi, ces câbles sont très facilement abîmés lors de l'installation des machines. Étant donné que le coût de ces câbles est élevé, il est important de trouver une solution technique ne nécessitant pas ce type de câbles.

La carte d'acquisition utilisée est programmée par un logiciel propriétaire. Ce logiciel cache la complexité des opérations et ne permet pas toujours de savoir comment optimiser ses développements. De plus, l'utilisation de ce type de logiciels limite le choix des fournisseurs à ceux qui supportent ce logiciel. Or, l'entreprise souhaite rester au maximum indépendante des fournisseurs de caméra et carte d'acquisition.

TABLE 1.2 – Caractéristiques de quelques standards de transport d'image

Standard	Débit maximal	Longueur max
GigE Vision	230 Mo/s	100 m
USB3 Vision	400 Mo/s	5 m
Camera Link	850 Mo/s	7 à 10 m
CoaXPress Quad	2500 Mo/s	40 à 100m (selon débit)

Une observation supplémentaire sur ce système de vision porte sur la communication carte d'acquisition-PC industriel. Cette communication permet de transmettre un débit de données allant jusqu'à 900 Mo/s. Cependant, le besoin actuel est de l'ordre de 10 Mo/s. Cette communication semble surdimensionnée.

Actuellement, il existe sur le marché des caméras avec pré-traitement intégré. La majorité de ces caméras sont basées sur des FPGAs. La détection de trait laser est une opération nécessitant un débit image important en entrée. Cependant, après détection du trait laser, le débit image est divisé par la hauteur de l'image. Effectuer cette opération au niveau de la caméra permettrait d'utiliser le capteur au maximum de sa vitesse. Le résultat en sortie ayant un débit plus faible, il est possible de s'affranchir de la limite de débit due aux câbles. Ainsi, il est possible d'utiliser des câbles et des protocoles de transfert d'images plus lents, plus économiques et plus robustes.

1.2 Objectifs

L'objectif principal de ce travail est de mettre en place un système de vision plus flexible, adaptable et non propriétaire.

Le système de vision à mettre en œuvre est représenté sur la figure 1.2. Il est constitué d'une caméra-FPGA programmable qui transmettrait le résultat de la détection de trait laser au PC industriel du système précédent. Le transfert du résultat se ferait via un protocole Gigabit Ethernet dont le débit est de 80 Mo/s. Cette communication est plus lente que le Camera Link mais comme le débit de données à transférer est plus faible, elle est pertinente.

Comme l'entreprise souhaite rester indépendante de ses fournisseurs, le choix de la méthode de développement du système de vision est important. Le développement FPGA est réputé difficile dans l'industrie. Notamment, les temps de développement, de debug et de qualification sont trop long dans un contexte industriel. C'est pourquoi la généricité de l'architecture matérielle développée est un point important de ce travail. L'objectif est de permettre à l'entreprise partenaire de réutiliser au maximum les développements et ainsi de transformer l'étape de portage de l'architecture matérielle d'une caméra à une autre en un problème d'interfaces au lieu d'un problème de développement. La généricité permet également d'envisager de nouveaux algorithmes. La suite de ce manuscrit montrera comment

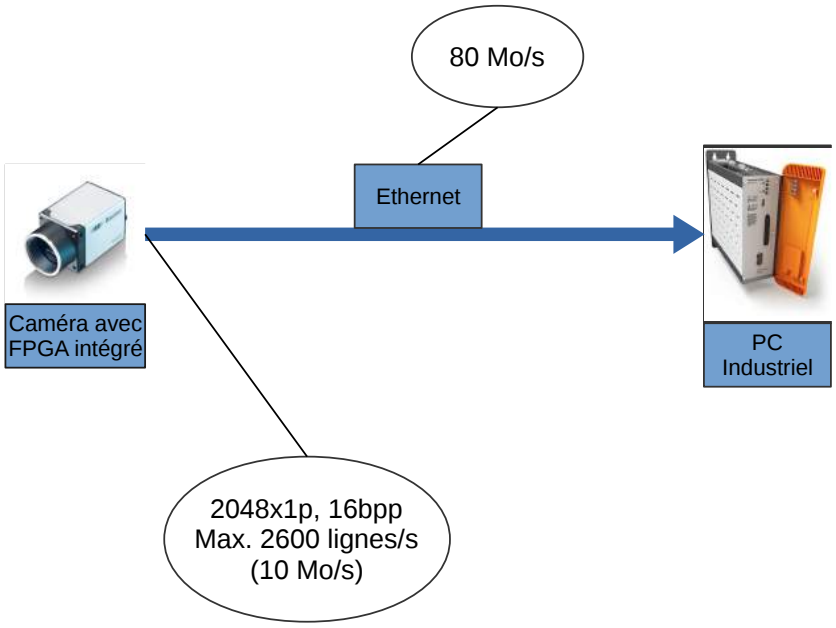


FIGURE 1.2 – Système de vision souhaité.

chacune de ces difficultés a été abordée.

1.3 Contributions

Dans un premier temps, il était important pour l'entreprise de savoir si l'algorithme utilisé dans le système de vision actuel était toujours d'actualité. La première contribution de cette thèse est donc un état de l'art des algorithmes de détection de trait laser. Ces algorithmes ont été développés et comparés suivant plusieurs critères qui sont : la précision, la justesse, le nombre d'opérations, la complexité algorithmique, la complexité mémoire, et le temps de calcul sur CPU. Ces études et comparaisons nous permettent de déduire quel type d'architecture (CPU, GPU, FPGA...) est le plus adapté à un algorithme de détection.

Une architecture matérielle générique a ensuite été définie pour l'algorithme de détection de trait laser utilisé à ce jour par l'entreprise. Cette architecture est adaptable aux évolutions du système de vision par l'entreprise, par exemple un changement de résolution de capteur. Cette IP a été testée et intégrée dans une caméra-FPGA standard du marché. Un prototype fonctionnel a été mis en place. Le système caméra-FPGA prototype a été testé sur une machine de test.

Le comportement de l'opérateur de détection de trait laser selon les valeurs des paramètres génériques a été étudié. Des métriques ont été fournies pour aider au dimensionnement du système. Ces métriques permettront d'aider les utilisateurs à faire évoluer le système vision.

1.4 Organisation du manuscrit

Ce manuscrit commence par une présentation du contexte de la thèse et de ses objectifs.

Le chapitre 2 présente la triangulation laser en général, ainsi que les algorithmes de détection de trait laser. L'analyse de ces algorithmes ainsi que leur comparaison seront également présentées.

Le chapitre 3 présente les FPGAs, leur flot de conception ainsi qu'une architecture générique pour système de vision sur FPGA.

Le chapitre 4 présente la réalisation de l'architecture générique pour l'opération de détection de trait laser. L'architecture de la caméra-FPGA utilisée sera présentée ainsi que les contraintes du système. La validation de l'opérateur de détection de trait laser sera également présentée dans ce chapitre.

Le chapitre 5 présente les dimensionnements effectués selon les paramètres de l'architecture. Le principe et l'intérêt d'effectuer ces dimensionnements seront également présentés.

Enfin, le chapitre 6 conclut ce manuscrit et présente les perspectives de ce travail.

Chapitre 2 : Triangulation laser

Historiquement, la triangulation est une méthode de mesure de distance utilisée depuis l'antiquité. Elle était principalement utilisée pour se repérer en mer ou tracer des cartes. La méthode consiste à se placer en deux points afin de former un triangle avec la cible. Puis, en chacun de ces points, mesurer l'angle entre la cible et l'autre point. Les deux angles obtenus, ainsi que la distance entre les deux points de vue, permettent de mesurer la distance avec la cible. C'est la triangulation.

La triangulation laser est une technologie utilisée pour le calcul de distance à partir d'une lumière structurée et d'un outil d'acquisition 2D qui est apparue à la fin des années 60 [For67]. L'objectif principal de sa mise en place était d'ajouter le sens de la vision à des robots manipulant des objets.

La figure 2.1 présente un système basique de triangulation laser. Les systèmes de triangulation laser sont principalement constitués (1) d'une source laser projetant un rayon, une ligne ou plusieurs lignes laser sur un objet, et (2) d'une caméra qui observe la projection du laser sur l'objet. À partir de l'angle et des distances entre la caméra et le laser, la distance entre la caméra et l'objet peut être obtenue par triangulation. Sur la figure 2.2 sont représentées des images acquises par le capteur image dans le cadre de la triangulation laser. La première image représente l'image acquise lorsqu'il n'y a pas d'objet sur le convoyeur. On peut voir une simple ligne laser. La seconde image représente le passage d'un objet sous le trait laser. On peut voir que le trait laser a une forme arrondi dans l'emplacement où se situe le produit. L'usage consiste à inverser l'image, la déformation du produit se fait donc vers le bas. C'est la déformation du trait laser par rapport à sa position d'origine qui donne la hauteur de l'objet. Lors du passage d'un produit, il est également fréquent qu'il y ait des zones où le trait laser est masqué et d'autres zones où le trait laser sur l'objet se superpose avec celui sur le convoyeur.

Aujourd'hui, la triangulation laser a de nombreuses applications dont les principales sont : la mesure de distance (télémètre laser), la détection d'obstacles, la numérisation 3D, et le contrôle qualité. Au niveau du système matériel, le principe reste le même sur la majorité des produits disponibles. Dans l'industrie, la triangulation laser est principalement utilisée pour la numérisation 3D d'objets pour du contrôle qualité et/ou de la manipulation d'objets par des robots.

Cependant la triangulation laser n'est pas la seule méthode d'acquisition tridimensionnelle existante. La section 2.1 de ce chapitre présente d'autres méthodes

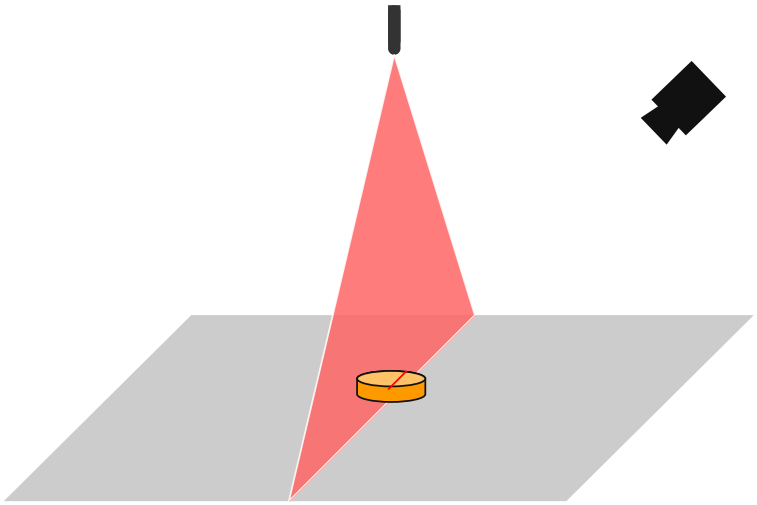


FIGURE 2.1 – Principe de la triangulation laser

(a) Image observée sans objet

(b) Image observée lorsqu'un objet passe sous le trait laser

FIGURE 2.2 – Images observées par la caméra dans un système de triangulation laser

d'acquisition tridimensionnelle et les raisons qui nous ont poussé à continuer d'utiliser la triangulation laser.

La section 2.2 est un état de l'art des systèmes de détection de trait laser, montrant les améliorations successives en termes de précision et de vitesse.

Depuis que la triangulation laser a été mise au point, de nombreux algorithmes ont été créés afin de détecter le laser avec une précision « sous-pixel ». Certains sont adaptés aux contraintes de l'industrie et des systèmes embarquées, d'autres moins. La section 2.3 présentera les principaux algorithmes de détection de trait laser existants.

La section 2.4 comprendra une comparaison des caractéristiques des principaux algorithmes de détection de trait laser, notamment en termes de précision et de complexité.

Enfin, la section 2.5 conclut ce chapitre.

2.1 Systèmes de vision pour l'industrie

Dans l'industrie, il est nécessaire de contrôler si la production est conforme à des caractéristiques préétablies, notamment, si la forme et les dimensions du produit sont conformes à un cahier des charges prédéterminé. Ce contrôle est de plus en plus souvent effectué par des systèmes de vision, notamment car ils ont l'avantage de ne pas nécessiter de contacts avec le produit. Un contrôle qualité sans contact ne détériore pas le produit et dans le domaine de l'agroalimentaire, ne risque pas de contaminer le produit. Deux catégories de systèmes de vision peuvent être utilisées pour cette inspection : les systèmes de vision 2D et les systèmes de vision 3D.

2.1.1 Les méthodes 2D

Ces systèmes de vision consistent principalement à mettre seulement une caméra au-dessus du convoyeur. C'est à partir des images acquises par cette caméra que sont contrôlés les produits. La caméra utilisée peut être matricielle ou linéaire, couleur ou monochrome.

Dans [FBR+10], les auteurs ont mis en place un système de vision pour faire du contrôle qualité dans le domaine de l'imprimerie. Le système est composé d'une caméra linéaire couleur avec FPGA intégré. Les auteurs ont décidé d'utiliser un capteur CMOS qui leur permet d'acquérir la région d'intérêt souhaitée en augmentant les performances en termes de vitesse. En effet, la technologie CMOS permet d'acquérir à une vitesse plus importante lorsque la région d'intérêt est réduite. Le traitement de l'image et le contrôle qualité sont effectués sur un FPGA couplé au capteur. Les auteurs ont pu acquérir et traiter des images acquises à une vitesse de 100 kHz.

Dans [ENG] est présenté un système de vision pour le contrôle qualité industriel exploitant l'apprentissage automatique. L'acquisition se fait à l'aide d'un capteur image matriciel. Les images sont traitées en interne et seul les résultats sont trans-

mis. Les inconvénients de ce produit sont un champ de vue encore trop faible pour notre besoin et la quantité relativement faible de produits traités par seconde (en moyenne 20 produits/s).

Les avantages de ces méthodes sont la simplicité d'installation et la possibilité d'effectuer tous les traitements à l'aide d'une seule caméra.

Souvent, les méthodes d'acquisition 2D utilisent la couleur pour analyser le produit. Dans un environnement farineux comme celui de la boulangerie industrielle, il est difficile de faire la différence entre le produit, généralement clair, et la farine, également claire, présente sur le convoyeur. Il faut donc éviter de se baser sur la couleur pour ce type de produit. C'est pourquoi les méthodes 3D sont privilégiées pour l'analyse de la forme du produit. Ces méthodes permettent également de connaître et contrôler la hauteur du produit.

2.1.2 Les méthodes 3D

Pour l'acquisition d'images 3D, plusieurs technologies existent : les méthodes avec contact et les méthodes sans contact. Dans le contexte de la boulangerie industrielle, ce sont les méthodes sans contact qui nous intéressent. Les principales méthodes sans contact sont :

- La stéréo-vision,
- La photogrammétrie,
- Le temps de vol,
- La projection de franges.

Ces méthodes sont présentées dans les sections suivantes.

2.1.2.1 La stéréo-vision

Une introduction à la stéréo vision est disponible dans [GSA10]. La stéréo-vision est basée sur le principe de la vision humaine. Le système est constitué de deux caméras placées sur le même plan épipolaire, qui acquièrent la même scène avec un point de vue très légèrement différent. Une correspondance est ensuite réalisée entre chaque point des deux images afin d'en déduire la distance jusqu'à l'objet ou les objets observés. Le résultat obtenu est une carte de disparité.

L'étape de correspondance est l'étape la plus complexe à réaliser et la plus consommatrice en termes de ressources et de temps de calcul. Un autre inconvénient est la difficulté voire l'impossibilité de faire la correspondance lorsque les textures des objets sont lisses et uniformes, ce qui peut mener à des erreurs de correspondance. Une étude des algorithmes de stéréo-vision et de leur pertinence par rapport à une exécution temps réel et embarquée est disponible dans [TLLA12]. L'exécution en temps réel de cette étape nécessite l'utilisation de plate-formes matérielles dédiées telles que celle présentée par [Mat13].

D'autres inconvénients de la stéréo-vision sont une précision qui dépend fortement de l'éclairage et une mesure présentant trop d'aléas. Malgré ces inconvénients, cette méthode a l'avantage d'être peu coûteuse en termes de matériel et de

fonctionner aussi bien en intérieur qu'en extérieur.

La stéréo-vision peut être utilisée pour des applications de détection d'obstacles chez des petits robots, de reconnaissance d'objets, ou de modélisation 3D.

Un exemple de produit basé sur la stéréo-vision est la caméra stéréo-vision couleur Bumblebee 2 de OMRON [Omr]. Elle acquiert les cartes de disparité à une vitesse de 48 images/s. Le produit [IDS] permet d'acquérir des cartes de disparité en temps réel avec une vitesse de 10 fps pour une résolution de 1280 x 1024, jusqu'à une distance de 3 m. Ce produit a été testé par l'équipe vision de l'entreprise et présentait des erreurs de mesure allant jusqu'à 5 mm sur certains pixels. Le produit présenté dans [MMC14] acquiert des cartes de disparité à une vitesse allant de 30 à 60 images/s pour une résolution allant de 640 x 480 à 752 x 480 pixels. Mis en place sur un FPGA Spartan 6 LX75, il a l'avantage d'avoir une très faible consommation (2 W), et d'être peu coûteux.

2.1.2.2 La photogrammétrie

La photogrammétrie est une méthode où les dimensions de l'objet observé sont obtenus à partir de plusieurs photographies du même objet prises sous des angles différents. Le principe est donc le même que pour la stéréovision. Les systèmes de photogrammétrie sont souvent constitués d'une caméra que l'on déplace dans un plan horizontal au-dessus de la zone à numériser. Dans [Sch94] sont présenté les concepts algorithmiques essentiels de la photogrammétrie. Dans [ZGWM14], les auteurs ont embarqué un système de photogrammétrie « presque » temps réel. La latence du système allait de 0.2 à 0.5 secondes selon l'application, avec une contrainte temps réelle d'une image à traiter toutes les 5 secondes. Cependant, des traitements supplémentaires sont réalisés après la fin de l'enregistrement afin d'augmenter la qualité et la lisibilité de la carte obtenue, ce qui fait que le système n'est pas totalement temps réel. Cette méthode d'acquisition 3D est utilisée pour la topographie ou encore dans le domaine de l'archéologie ou de la culture [CHT04]. Elle n'est pas pertinente dans un contexte industriel.

2.1.2.3 Les capteurs temps de vol

Le principe des capteurs temps de vol est basé sur la mesure du déphasage entre un signal émis et la réception de sa réflexion sur les obstacles qu'il rencontre. Une bonne introduction à la technologie temps de vol est le livre [HLCH12]. Un capteur composé de plusieurs cellules émet un ou plusieurs signaux infrarouge et chaque cellule de capteur reçoit une réflexion de ce signal. La différence de phase entre le signal et le signal émit permet d'obtenir la distance avec l'obstacle rencontré. Les principaux avantages de cette méthode sont :

- la simplicité et la compacité du système : par rapport aux systèmes présentés précédemment, il demande peu d'espace ;
- l'obtention directe d'une image de profondeur, idéal pour des applications temps réel ;

- à l'inverse de la stéréo-vision, une détection correcte sur des surfaces planes et lisses, et une moindre sensibilité à la lumière ambiante.

La précision de cette méthode est de 5 à 10 mm, ce qui est encore insuffisant pour notre contexte. Cette faible précision peut être due, entre autres, au flou causé par la mesure d'objets en mouvements [HLCH12], par une difficulté à détecter correctement le signal reçu sur les objets texturés (contrairement à la stéréo-vision), sur les bords et discontinuités des objets, ce qui cause des erreurs de mesure, ou par une horloge pas assez précise.

Il existe un grand nombre de produits « grand publiques » basés sur la technologie temps de vol aussi bien qu'industriels. Parmi les produits « grand publiques », la Kinect v2 est le plus populaire d'entre eux. Mais il existe également, le produit Xtion Pro live d'ASUS ou le produit Real Sense de chez Intel. En termes de produits industriels, il existe entre autres : [Hep], [IFM] et [Bas]. Dans l'industrie, ce type de scanner 3D est notamment utilisé dans la logistique.

2.1.2.4 La projection de franges

La projection de franges est une méthode d'acquisition 3D active pour laquelle est projeté sur la scène à capturer, une lumière structurée représentant des bandes noires (signal sinusoïdal) ou colorées. Une ou plusieurs caméras acquièrent cette scène, et les informations acquises sont traitées par une unité de calcul. Les franges sont analysées afin d'obtenir la phase du signal observé, à l'aide d'algorithmes basés entre autres sur la transformée de Fourier ou la transformée en ondelettes. La phase observée est déroulée puis convertie en distance. Un état de l'art sur la technologie de projection de franges, présentant les algorithmes et les applications de la méthode est disponible dans [GR10].

Les produits basés sur la projection de franges, sont souvent des outils de numérisation 3D pour le contrôle qualité de pièces mécaniques. Par exemple, les produits StereoScan neo et smartSCAN-HE de l'entreprise Eotech [EOT].

Cette méthode est en général utilisée pour scanner des pièces à l'arrêt et est donc peu compatible dans notre contexte où les produits avancent en continue sur le convoyeur.

2.1.3 Conclusion

Dans cette partie, des systèmes d'acquisition 3D pour l'industrie ont été présentés. Ces systèmes ne répondent pas actuellement aux besoins de l'entreprise qui sont : une précision élevé, un calcul temps réel et la nécessité de scanner des produits en mouvement sur un convoyeur. Pour toutes ces raisons, la triangulation laser est le système d'acquisition 3D majoritairement utilisé pour le contrôle qualité. La section suivante présente un état de l'art des systèmes de triangulation laser.

2.2 Systèmes de triangulation laser

[FS02], ont classé des articles portant sur la triangulation laser selon le type d'acquisition du signal. Les types d'acquisitions qu'ils ont rencontré sont : l'acquisition brut qui correspond à la majorité des méthodes, l'encodage spatial, et les capteurs intelligents. L'encodage spatial correspond à la projection de formes binaires sur la scène. Les méthodes dites "capteur intelligent", consistent par exemple à faire pivoter l'ensemble caméra-laser avec un timer au lieu d'un contrôleur externe. Cela permet de simplifier le système. Pour chaque article traité, la méthode de calibration utilisée a également été indiqué. Cet article se conclut sur le fait que la précision de la triangulation laser dépend essentiellement de la calibration et de la précision de la détection de trait laser.

Les premières applications de la triangulation laser concernent la robotique, notamment, la détection et reconnaissance d'objets, ainsi que leur préhension. Certains de ces systèmes sont basés sur la projection et la détection d'un point laser.

[For67] a mis en place un robot mobile équipé d'un télémètre laser. Ce télémètre est constitué d'un laser, d'un photomultiplicateur qui est un appareil permettant de capturer des photons (de nos jours équivalent à une caméra) et d'un miroir rotatif. Le laser projette un point laser sur un objet via le miroir rotatif qui est détecté par le photomultiplicateur. La rotation du miroir permet de déplacer le point laser sur l'objet. La distance entre le système et l'objet est obtenue via triangulation avec l'angle de rotation du miroir. Cette méthode est longue car il faut énormément de points pour réaliser l'acquisition 3D d'un objet.

La première manière d'accélérer le temps de calcul, est de capturer moins de points. [IN76] ont créé un système de vision permettant à des robots de reconnaître, saisir et déplacer des petits objets de façon précise, en acquérant un minimum de points. Le système de vision est constitué d'une source laser contrôlée par un ordinateur et d'une caméra tube dissecteur d'image. L'objectif de la caméra contient un filtre d'interférence (632,8nm) pour le laser. Les coordonnées de chaque point acquis sont obtenues via triangulation sur un ordinateur Melcom 7500 ou PDP 12. Les auteurs ont mis en place un protocole de reconnaissance d'objets qui classe les objets en trois groupes : brique ou cylindre, pyramide ou cone, et prisme. Ce protocole permet d'obtenir des caractéristiques telles que la hauteur, la largeur, la catégorie précise de l'objet et sa position. Les auteurs ont testé ce système avec un robot superposant des cylindres les uns sur les autres. Les résultats obtenus sont :

- Une erreur relative de 1.0% pour la coordonnée X et de 0.2% pour la coordonnée Y,
- Un temps de réponse de 45 μs pour l'outil d'acquisition 2D, mais un temps de réponse d'1 ms pour le contrôleur laser ce qui limite les performances du système,
- Le temps de réponse de ce laser traqueur est de 300 points/s.

Le principal défaut de ce système est sa consommation mémoire.

Une autre manière d'accélérer le temps de calcul est d'acquérir moins d'images mais avec plus d'informations. Pour cela, des systèmes basés sur la projection d'une ligne laser ont été mis en place. [Shi72] a créé un système de triangulation laser où l'on projette une ligne lumineuse pour gagner du temps sur l'acquisition de données. Ce système est utilisé pour reconnaître des polyèdres, et mesurer certaines de leurs caractéristiques tel que leur taille ou leur position. La principale problématique est de mettre en place une méthode d'acquisition 3D suffisamment fiable, précise et rapide. Ce système est constitué d'une source lumineuse placée sur une pièce mécanique rotative et d'une caméra TV acquérant les images. Le signal de l'image est envoyé à un préprocesseur qui convertit le signal analogique en signal numérique et divise l'image en sous-images de taille 256x256. Ces sous-images sont transmises à un ordinateur qui calcule les coordonnées de chaque point de l'objet par triangulation.Les premières applications de la triangulation laser concernent la robotique, notamment, la détection et reconnaissance d'objets, ainsi que leur préhension. Certains de ces systèmes sont basés sur la projection et la détection d'un point laser. Ensuite, les plans constituant les objets sont obtenus via des calculs de coûts. La forme et la position de l'objet sont retrouvées avec les équations de ces plans. Pour faire pivoter le projecteur dans un sens ou dans l'autre, l'ordinateur envoie une commande au contrôleur du projecteur. Il n'y a pas d'information sur le temps de calcul pour la détection de trait laser.

Enfin, la dernière manière d'accélérer les temps de traitement consiste à effectuer tous les calculs en même temps. C'est le cas pour les applications de type télémètre laser. [KGC91] ont créé un capteur intelligent qui capture les images et les traite en même temps. Ce capteur acquiert et détecte le trait laser de façon analogique par seuillage. Puis les résultats sont stockés sur chaque cellule de la puce le temps que l'objet soit entièrement scanné. Ensuite, ces résultats sont transmis à un PC via un convertisseur analogique-numérique. Le framerate obtenu varie de 100 à 1000 fps selon la luminosité de la scène et de l'objet (100 pour un objet sombre, 1000 dans des conditions optimales) pour une résolution de 28x32 pixels. Les principaux avantages de cette méthode sont une vitesse indépendante de la résolution, et la possibilité de garder un trait laser épais. Les principaux inconvénients sont une précision limitée à cause de l'algorithme de détection de trait laser choisi (la détection par seuillage ne permet pas une précision « sous-pixel »). Ce travail a été continué par [SYI94] qui y a ajouté un réglage automatique du seuil de détection. Ce réglage automatique permettrait de ne plus dépendre des conditions de luminosité de la scène. Le capteur développé parvient aux mêmes résultats en observant la même scène avec un éclairage fort ou faible avec une erreur inférieure à $120\mu m$ qui est l'erreur de quantification et la résolution théorique. Le trait laser est balayé à une vitesse de 30 Hz et la fréquence d'horloge d'un scan brut est de 30 kHz. Enfin, [OIA03] ont créé un capteur intelligent pour une application de type télémètre laser/numérisation d'objets basé sur le même principe. Le capteur développé a un framerate maximum de 260 kfps, ce qui correspond à 2034 images 3D de résolution 128x128 par seconde, ou bien à l'acquisition de 31.8 images 3D de résolution 1024x1024 par seconde. L'auteur a pu obtenir une précision « sous-pixel »

avec l'algorithme du centre de masse appliqué sur des pixels 3 bits.

Dans l'industrie, en plus de la vitesse, le principal défi est de rendre la détection de trait laser robuste au bruit, notamment aux réflexions de trait laser ou aux perturbations lumineuses. [CTC95] ont ajouté un système de polarisation à leur système de triangulation laser afin de limiter la réflexion du trait laser sur les surfaces spéculaires telles que les métaux. Un premier polariseur linéaire est placé au niveau du laser entre la lentille de focalisation et la "lentille à barre" (lentille cylindrique permettant de passer d'un point laser à un trait laser). Au niveau de l'acquisition d'images, les auteurs utilisent une caméra (caméra TM500 au format d'image HIPS) de résolution 512x768 pixels, 8 bits par pixels, avec une lentille de distance focale 25mm. Sur la caméra ont été ajouté : un filtre rouge pour couper les longueurs d'onde supérieures à 750 nm, un filtre de densité neutre (neutral density filter) pour éviter les effets de saturation sur le capteur CCD, et un second polarisateur linéaire. L'inconvénient de cette méthode est que le polarisateur doit être parfaitement aligné avec le laser : quelques pixels d'erreur peuvent diminuer sensiblement la précision du résultat. Les auteurs proposent comme solution d'utiliser des capteur de type "liquid crystal polarization camera".

L'industrie qui a fait le plus de recherches sur la détection de trait laser est la métallurgie et la soudure. Dans l'article écrit par [KSCK96], les auteurs ont eu besoin d'une correction en temps réel de la position de la jointure à souder dans un environnement industriel très bruité (reflets et fumées entre autres). Pour cela, un système de triangulation laser basé sur une caméra CCD capturant une image au format NTSC et un laser projetant une ligne laser a été créé. L'algorithme qu'ils ont développé est constitué de deux étapes qui agissent avant et pendant la soudure. La première étape permet d'identifier et de modéliser le joint. La seconde étape permet de détecter les caractéristiques du joint durant la soudure. Cette étape est soumise à des contraintes élevées de temps de calcul et de robustesse au bruit. [HP98] ont mis en place un système de triangulation laser low-cost pour la soudure dans l'industrie navale. Pour des raisons de temps de calcul et de robustesse au bruit, leur système est composé d'une caméra CCD qui envoie les images acquises analogiquement à une carte d'acquisition avec FPGA, où le FPGA va réaliser la détection de trait laser. Les résultats de la détection de trait laser sont envoyés par liaison RS485 à un PC industriel situé plus loin. Le capteur CCD utilisé a un débit image de 14 Mo/s. Les résultats de la détection de trait laser sont ensuite traités par le PC via un algorithme de logique flou servant à détecter la soudure. [ZZZG08] ont créé une méthode d'inspection de surface basée sur la triangulation laser pour le contrôle qualité des soudures laser. Le système matériel est composé d'un système optique (caméra CMOS, lentilles, filtre et laser), d'une unité de contrôle contenant une carte DSP qui traite l'image et une carte de contrôle qui extrait le profil du cordon de soudure et ses caractéristiques géométriques, enfin, les résultats sont envoyés à un PC qui analyse ces résultats. Le système de vision mis en place suit les étapes suivantes : lecture de l'image, filtrage médian, seuillage de l'image, ouverture, fermeture et extraction du milieu de la ligne laser. Les auteurs ont pu traiter les images pour des vitesses de soudure

à 8m par min. Il n'y a pas d'information sur la vitesse de traitement de l'image. [UMG10] ont mis en place une méthode d'extraction de ligne laser robuste au bruit pour l'industrie métallurgique. Les auteurs ont mis en place un protocole en plusieurs étapes : calibration avec calcul précis de la distance laser-objet et apprentissage de la position initiale du trait laser. Acquisition de l'image et calcul de la position du trait laser par l'algorithme centre de masse sur les zones où l'on s'attend à voir le trait laser. Lorsqu'on ne voit plus le trait laser dans la zone où il devrait être (le fond), le trait laser est recherché sur toute l'image. Ensuite, un post traitement en trois étapes est appliqué : segmenter la courbe mesurée, retirer les points aberrants, enfin, lissage du résultat. La méthode mise en place a été évaluée selon sa précision et son temps de calcul avec un processeur Intel Core2 Duo. L'erreur maximale obtenue est de 0.74 mm, le temps d'exécution moyen est de 1.44 ms. (600 fps pour une résolution de 640x260). [ZJY+12] ont mis en place un système de détection de lignes de soudure basée sur la projection d'une lumière structurée (laser) en forme de croix. Dans cette solution, le trait laser est détecté par l'application d'un filtre de Canny (détection de contour) puis d'un modèle de Markov caché. Avec leur système, ils traitent actuellement 20 images par seconde (images de résolution 500x400). Ils ont une erreur moyenne de 5 pixels.

D'autres industries ont également fait des recherches sur la détection de trait laser, notamment le contrôle qualité. Dans [ISIU99], les auteurs mettent en place un système de triangulation laser pour l'industrie avec une précision « sous-pixel ». Pour cela, les auteurs insistent sur le fait que chaque étape : calibration, détection de trait laser et reconstruction 3D, doit être réalisée avec une précision « sous-pixel ». Pour la détection de trait laser « sous-pixel », l'algorithme utilisé est le centre de masse. Les positions obtenues sont ensuite lissées par une interpolation polynomiale. Au niveau des résultats, ils ont obtenu une erreur de 0.2 pixel avec un écart-type de 0.099.

[Jol01] a mis en place un système de triangulation laser dans l'agroalimentaire pour la découpe industrielle de pièces de viande. Le système développé est basé sur des composants standards pour l'industrie. Cela comprend, une camera, une carte d'acquisition, et un PC industriel. La caméra acquiert les images, la carte d'acquisition les mets dans des buffers, puis les transmet au PC industriel qui effectue la détection de trait laser sur les images acquises, et la création d'images 3D et d'images d'intensité. D'après l'auteur, la vitesse de son système de vision a été limité par le temps de transfert de l'image et le temps pris par la détection de trait laser.

[LWZC09] ont mis en place un système de triangulation laser où le système caméra-laser se déplace et change l'inclinaison du laser selon la courbure de l'objet analysé. Les auteurs ont également décrit l'algorithme mis en place pour l'acquisition et le déplacement du système. Le problème rencontré par les auteurs étant que pour l'analyse de grands objets présentant des différences de hauteurs importantes (comme des pales d'avion par exemple), le champ de vue de la caméra doit être plus élevé. Cependant, un champ de vue élevé cause une perte de précision importante, les auteurs ont donc voulu créer un système avec un champ de vue

faible mais qui se déplace selon la forme de l'objet et corrige l'inclinaison du laser afin qu'il soit toujours perpendiculaire à la normale de la courbure de l'objet.

De nos jours, de nouvelles applications apparaissent régulièrement pour la triangulation laser, et permettent des innovations sur sa mise en œuvre.

[WCJ07] ont utilisé un système de triangulation laser pour mesurer les pulsations du pouls sur le poignet de patients. Le système matériel est constitué d'une diode laser et d'un capteur image CMOS. La détection du trait laser est effectuée par le calcul du centre de masse et la fréquence du pouls est calculée via une FFT.

[PFS12] ont mis en place un système de triangulation laser pour la détection et la préhension d'objets sous l'eau. Il ont choisi la couleur de leur laser suivant l'absorption de son spectre par l'eau. La détection de trait laser est faite en deux étapes : détecter le trait laser vert en calculant la distance euclidienne dans un espace RGB entre les pixels observées. Les pixels dont la distance euclidienne est inférieure à un seuil sont considérés comme appartenant au trait laser. Ensuite, la position « sous-pixel » du laser est calculée par centre de masse. Après détection du trait laser, les images 3D sont reconstituées puis les commandes d'un bras robotique sont calculées.

[BSF13] a mis en place un système de triangulation laser embarquée pour un outil d'assistance aux aveugles qui permettrait de détecter des obstacles. L'architecture qu'il a développé contient un capteur CMOS (format VGA - 660x492p), un laser, un microcontrôleur (ARM Cortex M3), et un moteur à vibrations. La détection du trait laser se fait via un seuillage couplé à la présence d'un filtre rouge. En termes de performances, une image est traitée en 2 ms, ce qui permettrait 500 images/s. Les auteurs ont décidé de traiter 30 images/s.

Enfin, les articles de recherche appliquée portent principalement sur l'amélioration de la précision de la triangulation laser (jusqu'au μm).

[PNM92] ont créé un télémètre temps réel ayant une précision pouvant atteindre 3 μm sur une profondeur de champ de 2 mm à une distance de 100 mm. Ce télémètre est composée d'un laser suivi d'un expanseur de faisceau, d'un masque optique et d'un réseau de diffraction. Ainsi, le point laser est divisé en plusieurs points laser qui convergent sur la même cible.

[CL95] ont traité le problème de la détection de trait laser sur les discontinuités, bords, angles ou changements de texture des objets scannés. Pour cela, ils ont mis en place un nouveau protocole de détection de trait laser basé sur une analyse spatio-temporelle. L'algorithme de la méthode est présenté section 2.3.

[FGIS05] ont mis en place un système de triangulation laser avec FOV variable. Un FOV variable permet d'avoir une mesure précise quelque soit la distance avec l'objet. La procédure mise en oeuvre consiste à faire l'acquisition avec un FOV faible (donc acquisition de ce qui est proche) dans un premier temps, puis à faire des acquisitions successives en augmentant régulièrement le FOV.

[QZZ+13] ont évalué les caractéristiques et appliqué l'algorithme de [Ste13] pour la détection de trait laser. L'algorithme de Steger consiste à convoluer la colonne contenant le trait laser par une gaussienne et à dériver le signal. La position du trait laser est donc le point de passage par zéro (entre le dernier point positif

et le premier point négatif) du résultat. Cet article analyse les relations entre la précision de la détection de trait laser, la qualité d'image et les caractéristiques du trait laser (puissance notamment). La taille optimale du noyau pour le lissage gaussien est également analysé. Enfin, les auteurs estiment la précision (variance) de la détection de trait laser à partir des conditions d'acquisition.

Le tableau 2.1 représente une synthèse des articles publiés concernant la triangulation laser en termes de vitesse, résolution et précision.

TABLE 2.1 – Récapitulatif

Référence	Résolution	Profondeur du pixel	Nombre d'images/s	Algorithme	Précision	Unité de calcul
[Shi72]				Seuillage	0.5 pixel	CPU
[IN76]					0.2 %	CPU
[KGC91]	28x32	1 bit	100 à 1000 i/s	Seuillage	0.5 pixel	Capteur/ASIC
[PNM92]					3 μm	CPU (co-processor)
[KSCK96]	640x480	8 bits	30 i/s	FIR (*dérivée 2de d'une gaussienne)	$\pm 0.5mm$	CPU TMS320C30 (33 MHz)
[HP98]	752x266	8 bits	$\sim$ 73 i/s	Centre de masse	0.1 mm	FPGA
[ISIU99]	752x582	(PAL)	25 i/s	Centre de masse	SD. 0.099 pixels	
[Jol01]	640x200	8 bits	130 Hz	Centre de masse		CPU
[OIA03]	128x128	3 bits	260 352 i/s	Centre de masse	0.1 pixel	Capteur/ASIC
[FGIS05]	640x480	(NTSC)	30 i/s	Filtre de Prewitt ou de Canny		CPU (MATLAB)
[WCJ07]	648x488	8 bits RGB	15 i/s	Centre de masse		CPU (MATLAB)
[ZZZG08]	768x576			Seuillage		DSP
[UMG10]	640x260		600 i/s	Centre de masse + Lissage	0.74 mm	CPU (Intel Core 2 Duo)
[ZJY+12]	500x400		20 i/s	Filtre de Canny + Modèle de Markov caché	Err. moy. 5 pixels	
[BSF13]	660x492		Max 500 i/s (choix de 30 i/s)	Seuillage		CPU (ARM Cortex M3)

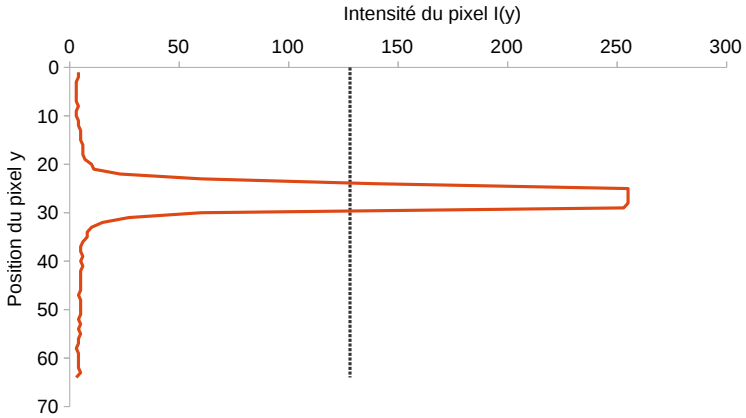


FIGURE 2.3 – Intensité des pixels sur une colonne d'image comprenant un trait laser

Dans cette partie, nous avons pu voir que les systèmes de triangulation laser sont utilisés dans des contextes applicatifs de plus en plus variés, et ayant des contraintes différentes. Ces contraintes différentes ayant entraîné des besoins différents, un nombre important de méthodes ont été mises en place pour la détection de trait laser. La section suivante présente les principaux algorithmes de détection de trait laser mis en place.

2.3 Algorithmes de détection de trait laser

Cette section a pour objet de présenter les algorithmes de détection de trait existants. La détection de trait laser consiste à retrouver pour chaque colonne ou ligne d'image, la position de la ligne laser sur l'image en étant le plus précis possible. La majorité du temps, en ayant une précision inférieure à 1 pixel (précision sub-pixellique). La figure 2.3 représente les pixels d'une colonne d'image comprenant un trait laser. Le pic d'intensité correspond au trait laser.

De nombreux algorithmes ont été mis en place pour la détection de trait laser. Actuellement, les principaux algorithmes peuvent être classifiés de la façon suivante :

- Les méthodes exploitant des seuils,
- Les méthodes exploitant des approximations et des interpolations,
- Le centre de masse,
- Les filtres fréquentiels à réponses impulsionnelles à support temporel fini (RIF ou FIR en anglais),
- Les méthodes statistiques.

La suite de cette section présente les principaux algorithmes rencontrés.

Dans un cas idéal, l'opération s'arrête après le calcul de la position du trait laser. En réalité, il est fréquent d'avoir des superpositions de lignes laser dues à

des réflexions ou à la géométrie de l'objet notamment lorsque l'objet a une forme courbe ou irrégulière. Il faut donc souvent ajouter une étape où l'on détermine quel est le trait le plus susceptible de correspondre au produit observé avant, après ou pendant le calcul. S'il y a superposition, il y a également des cas où le trait laser est absent de l'image. Dans ce cas, le résultat le plus approprié est 0.

2.3.1 Le centre de masse

La méthode du centre de masse a été mise au point par [JC81]. La formule du centre de masse sur une colonne d'image est rappelée dans les équations 2.1 à 2.3.

$$S_1 = \sum_{j=y_{min}}^{y_{max}} I(j) \tag{2.1}$$

$$S_2 = \sum_{j=y_{min}}^{y_{max}} I(j) \times j \tag{2.2}$$

$$y_{peak} = \frac{S_2}{S_1} \tag{2.3}$$

avec :

- $I(j)$ l'intensité du pixel de la ligne j,
- y_{min} et y_{max}, 2 bornes situées avant et après le trait laser,
- y_{peak}, la position du trait laser avec une précision sub-pixellique.

Pour pouvoir être précis, le calcul doit être appliqué près du pic d'intensité de l'image. C'est pourquoi, on couple souvent cette méthode avec un seuil ou une recherche du maximum. Cette méthode est l'une des plus utilisées et est encore appliquée aujourd'hui. [HP98, Jol01].

Les bornes y_{min} et y_{max} peuvent être choisies soient comme étant les premières et dernières valeurs supérieures à un seuil, soit comme étant la N^{ime} valeur avant (pour y_{min}) et après (pour y_{max}) le maximum d'intensité.

2.3.2 Les FIRs

Les méthodes par filtre FIR ont été introduites par [BR86]. Cette méthode consiste à filtrer chaque colonne ou ligne de l'image avec un filtre passe-bande, puis de calculer le point de passage par 0 du signal filtré. Cette méthode a l'avantage de réduire le bruit et de permettre le calcul de la position du trait laser avec une précision « sous-pixel ». Les filtres choisis par [BR86] sont un filtre d'ordre 4 et un filtre d'ordre 8 dont les coefficients sont donnés par les équations 2.4 et 2.5.

$$H_1(z) = 1 + z^{-1} - z^{-3} - z^{-4} = H_1'(z)(1 - z^{-3}) \tag{2.4}$$

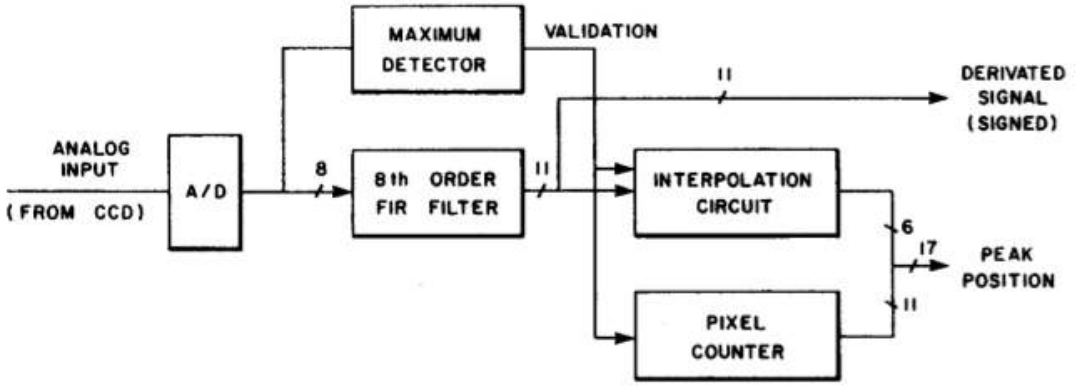


FIGURE 2.4 – Système de détection de trait laser mis en place par [BR86].

$$H_2(z) = 1 + z^{-1} + z^{-2} + z^{-3} - z^{-5} - z^{-6} - z^{-7} - z^{-8} = H_2'(z)(1 - z^{-5}) \quad (2.5)$$

avec $H_1'(z) = (1 - z^{-1})$ et $H_2'(z) = 1 + z^{-1} + z^{-2} + z^{-3}$.

Le filtrage FIR d'ordre 8 est celui qui donne les meilleurs résultats en termes d'erreur quadratique moyenne. La réalisation pratique du système de [BR86] est représentée sur la figure 2.4.

[KSCK96] ont également utilisé un filtrage pour la détection de trait laser sur leur système. Les auteurs ont décidé de filtrer chaque colonne de l'image par la dérivée seconde d'une gaussienne dont la largeur correspond à 3 fois la largeur du trait laser en pixels. La position de la valeur maximale du signal filtré (s'il est supérieur à un seuil donné) correspond à la position du trait laser.

$$R(j) = \sum_{j=-3/2L_w}^{3/2L_w} w_j I(y_0 + j) \quad (2.6)$$

$$R(j_{peak}) = \max(R(j)) \quad (2.7)$$

$$y_{peak} = \begin{cases} j_{peak} & \text{si} \quad R(j) > T \\ 0 & \text{sinon} \end{cases} \quad (2.8)$$

avec $I(j)$ le pixel situé à la ligne j, L_w un nombre pair représentant la largeur approximative du trait laser en nombre de pixels, w_j le $j$ième coefficient du filtre, y_0 la position autour de laquelle est appliquée ce filtre (souvent la ligne correspondant au pixel d'intensité maximal), $R(j)$ le signal filtré, j_{peak} la position correspondant au max du signal filtré, y_{peak} la position du trait laser et T un seuil prédéfini au-dessus duquel on considère le signal comme étant un trait laser. Avec $L_w = 4$, un exemple de coefficients pour ce filtre peut être : [-1 ; -1 ; -1 ; -1 ; 2 ; 2 ; 2 ; 2 ; -1 ; -1 ; -1 ; -1 ; -1].

[FSCP04] ont également créé une méthode de détection de trait laser exploi-

tant un filtrage FIR. Les auteurs sont partis du principe que le trait laser a un profil gaussien, puis montrent que filtrer la colonne d'image avec un filtre dont les coefficients correspondent à une dérivée de gaussienne permet d'appliquer un filtre passe-bande et de trouver la position du trait laser en calculant le point de passage par 0 du signal filtré. Les coefficients du filtre peuvent être calculés en appliquant la formule suivante :

$$g(x) = \frac{x}{\sqrt{2\pi}\sigma^3} \exp\left(\frac{-x^2}{2\sigma^2}\right) \tag{2.9}$$

Le point de passage par 0 entre les points de coordonnée $(y_A; I'_A)$ et $(y_B; I'_B)$ est défini par la formule suivante :

$$y_{peak} = y_A - \frac{I'_A.(y_B - y_A)}{I'_B - I'_A} \tag{2.10}$$

Cette méthode est également celle proposée par [Ste13] pour la détection de profils paraboliques et gaussiens. [Ste13] utilise en plus la dérivée seconde pour confirmer la position calculée.

La principale difficulté du filtrage FIR pour la détection du trait laser réside dans le fait de choisir la bonne fréquence de coupure.

2.3.3 Les approximations et interpolations

La recherche de la position du trait laser par approximation ou par interpolation consiste à supposer que le profil de trait laser a une certaine forme paramétrique (soit gaussienne, soit linéaire, soit parabolique, ...), et à obtenir la position du maximum de la courbe paramétrée. Ces méthodes appliquées en utilisant 3 points ont été détaillées par [FKN01]. Elles ont l'inconvénient de ne pas pouvoir être appliquées en cas de trait laser saturé comme représenté sur la figure 2.3. Les approximations gaussiennes et paraboliques sont les plus fiables.

Une formule généralisée de l'approximation gaussienne permettant d'obtenir la position du trait laser autour de la colonne y_0 avec $2P + 1$ pixels à traiter est :

$$y_{peak} = y_0 + \delta \tag{2.11}$$

avec y_0 la ligne autour de laquelle est calculée la position du trait laser (il s'agit souvent de la ligne d'intensité maximale), et δ définie de la manière suivante :

$$\delta = \frac{2P+1}{6} \times \frac{\sum_{j=1}^{P} (f(y_0 - j) - f(y_0 + j))}{\sum_{j=1}^{P} (f(y_0 + j) + f(y_0 - j)) - 2Pf(y_0)} \tag{2.12}$$

avec :

$$f(y) = ln(I(y)) \tag{2.13}$$

avec toujours $I(y)$ correspondant à l'intensité lumineuse sur l'image à la ligne y.

2.3.4 La méthode statistique

[SCL15] ont créé une méthode de détection de trait laser pour des lasers ayant une distribution uniforme. Cette méthode est basée sur le calcul des moments statistiques autour de la position du trait laser. Cela nécessite que l'image soit centrée autour du trait laser.

La première étape consiste à calculer les moments statistiques d'ordre 1, 2 et 3 autour de la position du trait laser.

$$\bar{m_i} = \frac{1}{n}\sum_{j=1}^{n} I(j)^i, i = 1, 2, 3 \tag{2.14}$$

avec, n le nombre de pixels dans la colonne, $I(j)$ l'intensité du pixel de la ligne j, et i l'ordre du moment statistique.

La seconde étape consiste à calculer le moment statistique d'ordre 1 sur la première moitié de l'image choisie, en s'assurant que le bord du trait laser en fait parti.

$$\bar{m_1}' = \frac{1}{n}\sum_{j=1}^{\frac{n}{2}} I(j) \tag{2.15}$$

Ensuite, il suffit de calculer les intensités moyennes de trait laser h_1 (pour le fond) et h_2 (pour le trait laser), ainsi que leur proportion (ou probabilité) respectives p_1 et p_2.

$$\begin{cases} h_1 = \frac{-b-\sqrt{b^2-4ac}}{2a} \\ h_2 = \frac{-b+\sqrt{b^2-4ac}}{2a} \\ p_1 = \frac{\bar{m_1}-h_2}{h_1-h_2} \\ p_2 = 1 - p_1 \end{cases} \text{ avec } \begin{cases} a = \bar{m_2} - {\bar{m_1}}^2 \geq 0 \\ b = \bar{m_1} \times \bar{m_2} - \bar{m_3} \leq 0 \\ c = \bar{m_1} \times \bar{m_3} - {\bar{m_1}}^2 \geq 0 \end{cases} \tag{2.16}$$

La position du trait laser correspond à la position du bord du trait laser auquel on ajoute la moitié de la largeur du trait laser (p_2).

$$y_{peak} = n \times \left(\frac{\bar{m_1}' - h_2}{2 \times (h_1 - h_2)} + \frac{p_2}{2} \right) \tag{2.17}$$

2.3.5 L'analyse spatio-temporelle

L'analyse spatio-temporelle est une procédure mise en place par [CL95] pour traiter le problème de la détection de trait laser sur les discontinuités, bords, angles ou changements de texture des objets scannés.

Le système matériel utilisé est composé d'un scanner Cyberware MS platform. Il s'agit d'un scanner classique de triangulation laser composé d'un laser projetant

un point laser et de deux caméras CCD, positionnées avec un angle de 30 ° par rapport au plan laser. Étant donné que les auteurs ont besoin de l'image non traitée des caméras pour tester leur méthode, un enregistreur vidéo Abekas A60 a été ajouté au système. L'image acquise a une résolution de 486x720 et la fréquence d'acquisition est de 30 Hz.

Une image spatio-temporelle est une image dont les colonnes correspondent au balayage d'une zone précise de l'image selon le temps. La méthode consiste à ne pas traiter les colonnes indépendamment, mais à prendre en compte le fait que le laser, les produits, et la caméra ne sont pas alignés.

Le processus de traitement de l'image est le suivant : (1) scanner l'objet et acquérir les images spatio-temporelles (images + timestamp) ; (2) Faire pivoter les images spatio-temporelles par l'angle $-\theta$ qui est l'angle entre le laser et le capteur image ; (3) en supposant que le trait laser a un profil gaussien, calcul des statistiques de la Gaussienne telles que la moyenne, la dispersion (écart-type) et l'amplitude dans les coordonnées pivotées pour chaque image acquise ; (4) transformation vers les coordonnées d'origine (rotation par θ). Cette méthode de détection de trait laser permet de réduire les distorsions causées par les discontinuités dans la couleur et la forme de l'objet. L'influence du bruit de Speckle est réduite.

2.3.6 Conclusion

Les principaux algorithmes de détection de trait laser ainsi que leur principe ont été présentés dans cette partie. Un nombre important d'algorithmes existent pour la détection de trait laser, mais il est difficile de savoir lequel utiliser dans un contexte donné. Le centre de masse a été très populaire jusqu'à récemment. De nos jours, le filtrage FIR semble être l'algorithme le plus couramment utilisé. Il est intéressant pour l'entreprise de savoir quel algorithme correspond le mieux à son contexte applicatif, et donc de comparer ces algorithmes entre eux. La section suivante présente une comparaison que nous avons effectuée entre plusieurs algorithmes présentés dans cette section.

2.4 Comparaisons des algorithmes

Plusieurs publications ont comparé des algorithmes de détection de trait laser.

[FKN01] ont comparé la précision « sous-pixel » de 5 algorithmes de détection de trait laser qui sont : l'approximation gaussienne, le centre de masse, l'interpolation linéaire, l'estimateur parabolique et les détecteurs de [BR86]. Ils ont utilisé l'erreur maximale, le biais, la robustesse au bruit et le comportement en cas de saturation. Selon leur étude, les meilleurs algorithmes sont l'approximation gaussienne, le centre de masse avec 7 points, et le détecteur de Blais et Rioux [BR86]. Les méthodes *centre de masse* et *détecteur de Blais et Rioux* sont précises mais sensibles au bruit, notamment aux réflexions du laser autour du pic. L'approximation gaussienne n'est pas sensible aux réflexions du laser, mais a l'inconvénient

d'avoir le temps de calcul le plus élevé.

[HP98] ont également comparé dix algorithmes de détection de trait laser pour choisir celui qui répondait le plus à leurs besoins exprimés en termes de précision, vitesse et robustesse aux réflexions de trait laser. La recherche du maximum est la méthode la plus rapide mais elle est peu précise. Le centre de masse est l'algorithme qui est le plus précis, avec un temps de calcul dans la moyenne. L'approximation gaussienne est moins précise et a un temps de calcul plus élevé. Afin de résoudre les erreurs liées aux réflexions du laser, les auteurs ont décidé de combiner la recherche du maximum (pour le temps de calcul) et la méthode du centre de masse : on calcule le centre de masse autour du maximum.

La comparaison des algorithmes de détection de trait laser la plus récente a été faite par [FSCP04]. Les auteurs ont comparé 6 algorithmes qui sont : le filtrage FIR par une dérivée de gaussienne, l'approximation gaussienne, le centre de masse, l'approximation linéaire, le détecteur de Blais et Rioux, et l'estimateur parabolique. Les tests ont été effectués sur des surface opaques et des surfaces translucides et avec plusieurs niveaux de rapport signal sur bruit (S/N). Les résultats obtenus sont les suivants :

- Dans les cas où le trait laser est étroit et peu intense, le filtrage FIR est celui qui montre les meilleurs résultats.
- Dans le cas des traits laser larges, le centre de masse est le meilleur choix d'algorithme.
- Dans le cas des surfaces translucides, le filtrage FIR est meilleur lorsque l'intensité du laser est élevée et le centre de masse est meilleur lorsque l'intensité du laser est faible. Le bruit de Speckle est une limite importante à la détection de trait laser dans ce cas.
- Dans tous les cas, le filtrage FIR est meilleur que l'algorithme de Blais et Rioux.

Cette comparaison a été réalisée dans un contexte applicatif différent du nôtre. Les auteurs n'ont par exemple pas pris en compte des considérations liées au développement de ces algorithmes sur système embarqué. Nous avons donc décidé d'effectuer notre propre analyse. Nous avons décidé de comparer sur CPU les algorithmes présentés section 2.3 selon des critères qui sont :

- la complexité algorithmique,
- la précision et le biais,
- le temps d'exécution,
- la consommation mémoire.

Les algorithmes et leur variantes que nous avons choisi sont les suivants :

- recherche du premier maximum et retour de sa position (*Max*),
- recherche du maximum et retour de la position moyenne du maximum si plusieurs maximums locaux (*Moy*),
- recherche du maximum et retour de la position médiane du maximum si plusieurs maximums locaux (*Med*),

- calcul du centre de masse avec 2N+1 points centrés sur le maximum (CM),
- détecteur de [BR86] avec 2N+1 points centrés sur le maximum (BR),
- approximation gaussienne avec 2N+1 points centrés sur le maximum (AG),
- moments statistiques avec 2N+1 points (MS),
- filtrage FIR par une dérivée de Gaussienne avec 2N+1 points (DG).

Chacun de ces algorithmes ont été développés en plusieurs versions :

- au format entier, flottant ou double,
- en utilisant l'image entière ou seulement les pixels supérieurs à un seuil.

2.4.1 Complexité algorithmique

L'étude de la complexité algorithmique consiste à évaluer le nombre d'opérations nécessaires à la réalisation de l'algorithme selon les paramètres de l'algorithme. Dans le cas de la détection de trait laser, ces paramètres peuvent être la largeur du trait laser en pixels ou le nombre de points utilisés. Il peut également s'agir de la hauteur d'image.

2.4.1.1 Coût des opérateurs sur FPGA

Les opérateurs arithmétiques sont plus ou moins coûteux à implanter sur FPGA. Le guide utilisateur de l'outil XST de Xilinx présente les opérations arithmétiques qui sont supportées par l'outil de synthèse. Par « supportées », cela signifie que ces opérateurs ont leur propres composants associés sur le FPGA. Les principaux opérateurs supportés sont :

- additionneurs, soustracteurs, et additionneur/soustracteurs,
- comparateurs,
- multiplicateurs (classiques, séquentiels ou pipelinés),
- les opérations multiplication - addition/soustraction et les opérations multiplication - accumulation (MACs) qui correspond à l'enchaînement de registres, multiplicateurs et additionneur/soustracteurs,
- et les divisions par une puissance de 2.

Les autres opérateurs doivent donc être conçus par le designer. Dans notre contexte, ces autres opérateurs sont : division entre 2 variables, logarithme népérien, racine carré, exponentielle et autres opérateurs trigonométriques : cosinus, sinus et tangente entre autres.

La plupart de ces opérateurs peuvent être obtenus avec des algorithmes itératifs et séquentiels, ou par l'utilisation de LUTs. Les algorithmes itératifs/séquentiels consistent à calculer un bit du résultat par itération, en faisant une soustraction ou une addition suivi d'un décalage à chaque itération. L'utilisation de LUTs consiste à mettre dans une ROM les résultats de l'opérateur pour les variables d'entrée qui nous intéressent. Cette méthode est intéressante lorsque la variable d'entrée a un faible nombre de bits, mais elle demande énormément de ressources lorsque la taille de la variable d'entrée augmente.

TABLE 2.2 – Opérateurs arithmétiques et leur classe d'implémentation sur FPGA

Classe d'implémentation	Opération
d0	$+$, $-$, $\pm$, $<$, $>$, $\geq$, $\leq$, $=$, $\times$, MAC, $(\div 2^n)$, $(\div Constante)$
d1	$\div$, $\surd$
d2	$exp()$, $ln()$, cos, sin, tan, etc.

Nous avons décidé de classer les opérateurs nécessaires à la réalisation de ces algorithmes en classes. La classe d0 représente les opérations supportées nativement par les outils de synthèse FPGA. La classe d1 comporte les opérateurs réalisables par des algorithmes naïfs sans condition sur les variables en entrée. La classe d2 comporte les opérateurs réalisables par des algorithmes imposant des conditions sur les variables en entrée (zone de convergence notamment) ou imposant l'utilisation de tables.

Pour la division, [OF97] ont présenté et comparé 5 algorithmes existants en 1997. Parmi ceux-là, les plus connus sont l'algorithme SRT, l'algorithme de Newton-Raphson, et l'algorithme de Goldschmidt.

La méthode CORDIC (pour *COordinate Rotation DIgital Computer*) dont une description est disponible dans [Mul06] permet de calculer efficacement la plupart des opérateurs arithmétiques sur FPGA, notamment les opérateurs trigonométriques. Les opérateurs racine carrée, logarithme et exponentiel peuvent être obtenus en combinant plusieurs algorithmes CORDIC.

Enfin, [Det07] a mis en place la méthode HOTBM (pour *Higher-Order Table-Based Method*) permettant de créer des opérations arithmétiques en virgule flottante sur FPGA et a pu obtenir de meilleurs performances sur FPGA que sur CPU avec ces opérateurs.

2.4.1.2 Nombre d'opérations nécessaire à chaque algorithme

Nous avons compté le nombre d'opérations nécessaires pour chacun des algorithmes choisis. Le nombre d'opérations des algorithmes de détection de trait laser dépend de variables qui sont le nombre de pixels dans une colonne (c'est-à-dire le nombre de lignes de l'image), et le nombre de points sur lesquels sont effectués les calculs. Pour l'algorithme basé sur les moments statistiques, on supposera que le calcul est effectué sur toute la colonne.

En notant N le nombre de lignes de l'image et L le nombre de pixels sur lesquels sont effectués les calculs, on peut estimer le nombre d'opérations nécessaires pour chaque algorithme selon ces paramètres. Ces estimations sont présentées dans le tableau 2.3. Des estimations sont visibles dans la table 2.4, en choisissant des couples de valeurs (N, L).

Pour la majorité des algorithmes, l'évolution du nombre d'opérateurs nécessaires de classe d0 est linéaire selon N et L, ou constant. Les seules exceptions sont

TABLE 2.3 – Nombre d'opérations pour chaque algorithme

Algorithme	Nb. op. classe d0	Nb. op. classe d1	Nb. op. classe d2
Max	$N-1$ comparaisons		
Med	$N-1+env.3L$ comparaisons		
Moy	$N-1$ comparaisons, $L-1$ additions	1 division	
CM	N comparaisons $2(L-1)$ additions, $2L$ multiplications	1 division	
BR, *DG*	$N-1$ comparaisons, $(L-1).(N-1)$ additions/soustractions, 3 soustractions, 1 multiplication	1 division	
AG	$N-1$ comparaisons, 2 multiplications, $(L-1)/2+1$ soustractions, $(L-1)/2-1+L$ additions	1 divisions	$2L-1$ logarithmes
MS (h_1, h_2, p_1, p_2 connus)	$(N-1)/2+1$ additions, 2 soustractions, 2 multiplications	3 divisions	

TABLE 2.4 – Nombre d'opérations pour chaque algorithme

(N, L)	*Max*	*Med*	*Moy*		*CM*		*BR*, *DG*		*AG*			*MS*	
	d0	d0	d0	d1	d0	d1	d0	d1	d0	d1	d2	d0	d1
(64, 5)	63	78	64	1	84	1	319	1	74	1	9	37	3
(64, 7)	63	84	69	1	90	1	445	1	78	1	13	37	3
(64, 9)	63	90	71	1	98	1	571	1	82	1	17	37	3
(128, 5)	127	142	131	1	146	1	639	1	138	1	9	69	3
(128, 7)	127	148	133	1	154	1	893	1	142	1	13	69	3
(128, 9)	127	154	135	1	162	1	1147	1	146	1	17	69	3

les algorithmes *BR* et *DG*, dont l'augmentation du nombre d'opérateurs de classe d0 est quadratique ($N \times L$). Pour les opérateurs de classe d1, le nombre d'opérations est constant pour tous les algorithmes. Seul l'algorithme *AG* nécessite une opération de classe d2, l'augmentation du nombre d'opérations pour cet algorithme

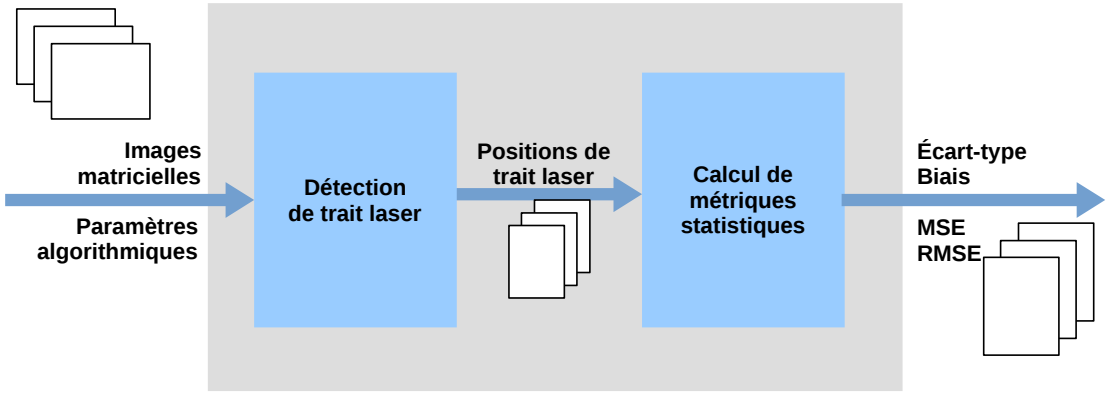


FIGURE 2.5 – Plate-forme logicielle d'évaluation des algorithmes de détection de trait laser

est linéaire par rapport à L. Les algorithmes *Max* et *Med* sont les seuls nécessitant seulement des opérateurs de classe d0. Les algorithmes *CM* et *MS* nécessitent des opérateurs de classe d0 et d1, mais nécessitent peu d'opérateurs de classe d0. L'algorithme *AG* nécessite des opérateurs de classe d0, d1 et d2. Cependant, il nécessite peu de chacun de ces opérateurs. Les algorithmes *BR* et *DG* nécessitent un nombre important d'opérateurs de classe d0 et 1 opération de classe d1.

2.4.2 Précision des algorithmes

Une plate-forme logicielle a été mise en place afin de comparer les principaux algorithmes de détection de trait laser. Cette plate-forme a pour objectif de calculer des métriques statistiques pour chacun des algorithmes étudiés.

2.4.2.1 Présentation de la plate-forme

Le principe de la plate-forme est représenté figure 2.5. Elle a été développée en C++ pour la partie calculant la position du trait laser et les résultats ont été exploités sur Matlab. Cette plate-forme traite une image en prenant en compte certains paramètres des algorithmes, tels que le nombre de points à prendre en compte pour le calcul de la position du centre de masse ou le seuil de détection laser par défaut. La plate-forme applique ensuite chacun des algorithmes choisis ainsi que leurs variantes sur l'image d'entrée. Les résultats qui sont la position du trait laser, la hauteur associée, et le temps de calcul en μs, sont écrits dans des fichiers textes pour chacun des algorithmes. Les résultats contenus dans ces fichiers sont ensuite traités afin d'en extraire des métriques statistiques. Ces métriques statistiques serviront de base à la comparaison des algorithmes.

2.4.2.2 Choix des métriques

Afin de comparer les algorithmes, nous avons choisi les métriques suivantes :

FIGURE 2.6 – Banc de test mis en place

FIGURE 2.7 – Mire utilisée

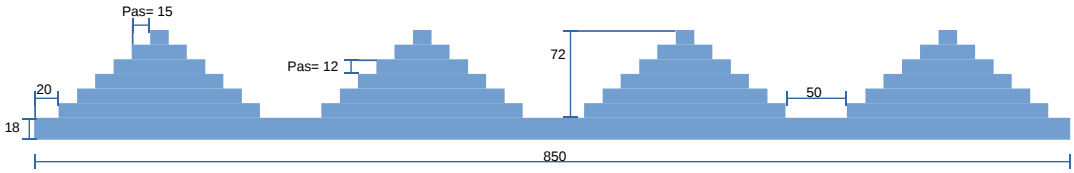


FIGURE 2.8 – Dimensions de la mire utilisée

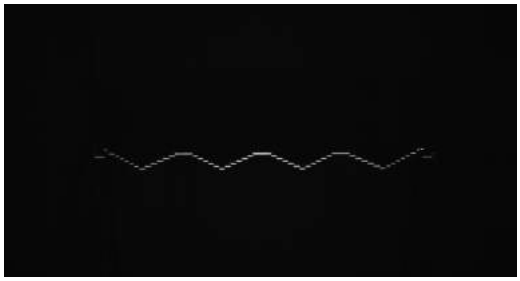
- L'écart-type qui permet d'estimer la précision des résultats,
- Le biais qui permet d'estimer la justesse du résultat,
- L'erreur quadratique moyenne qui est la somme des deux informations précédentes au carré.
- Le taux de succès qui estime la proportion de trait laser trouvé par un algorithme.

2.4.2.3 Images testées

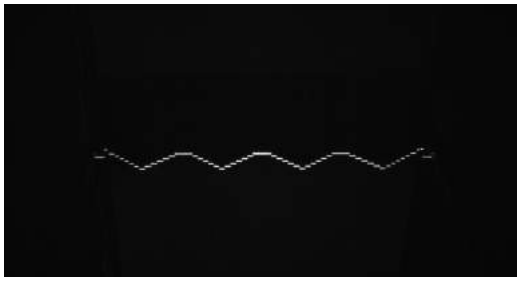
Les images utilisées lors de cette comparaison sont des images réelles acquises par une caméra-FPGA sur un banc de test, sans qu'aucun traitement ne soit réalisé sur les images. Le banc de test, représenté figure 2.6, est composé de la caméra-FPGA sur laquelle a été ajouté un filtre rouge, d'un laser et d'un objet qui nous sert de vérité terrain. La vérité terrain est la mire de calibration du système de vision utilisée par l'entreprise. Il s'agit d'une pièce en inox constituée de quatre pyramides ayant des paliers avec un pas de 12 mm. Elle est représentée sur la figure 2.7 et ses dimensions sont visibles sur la figure 2.8.

Afin d'étudier l'influence de deux types de bruits qui sont le bruit de Speckle et la saturation du capteur, nous avons capturé les images avec des temps d'exposition variant de 50 à 1000 μs. Lorsque le temps d'exposition est faible, le trait laser a une intensité plus faible et le bruit de Speckle est plus visible. À l'inverse, lorsque le temps d'exposition est élevé, le trait laser a une intensité plus forte sur l'image et

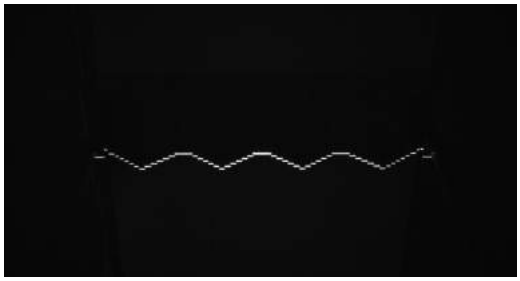
(a) 100 μs



(b) 200 μs



(c) 400 μs



(d) 500 μs

FIGURE 2.9 – Images acquises avec des temps d'exposition différents

est saturé. La figure 2.9 représente des images de la mire acquises avec différents temps d'exposition. Comme l'indique l'usage, la mire est dirigée vers le bas sur l'image.

Sur les images, nous pouvons voir de chaque côté de la mire, les supports et ridelles, ainsi qu'une petite portion de bande. De plus, l'intensité du trait laser est différente selon sa position sur l'image. Cela est dû à la non uniformité des lasers, ainsi qu'à des défauts au niveau de l'objectif qui font que la lumière captée sur les côtés est moins intense qu'au centre de l'image. De plus, le matériau ayant permis la réalisation de la mire n'est pas tout à fait mat ; les réflexions du laser à certains endroits de la mire font que le trait laser est plus intense au centre de l'image.

2.4.2.4 Résultats

La plate-forme a été utilisée avec les images présentées précédemment. Les résultats en termes d'écart-type, de biais, d'erreur quadratique moyenne et de taux de succès sont visibles sur les figures 2.10 à 2.13.

Globalement, tous les algorithmes ont une précision (écart-type) comprise entre 0.2 et 0.6 mm, sauf l'algorithme *Max* qui renvoie la position du premier maximum trouvé. Si l'on compare les écarts-types moyens de chaque algorithme pour notre lot d'images, l'algorithme le plus précis est l'algorithme *DG*, suivi de l'algorithme *CM* et de l'algorithme *MS*. Pour les algorithmes *Max*, *DG*, *BR* et *AG*, l'écart-type augmente lorsque le temps d'exposition augmente. Cela signifie que ces algorithmes perdent en précision lorsque le trait laser est saturé, et sont donc plus précis pour

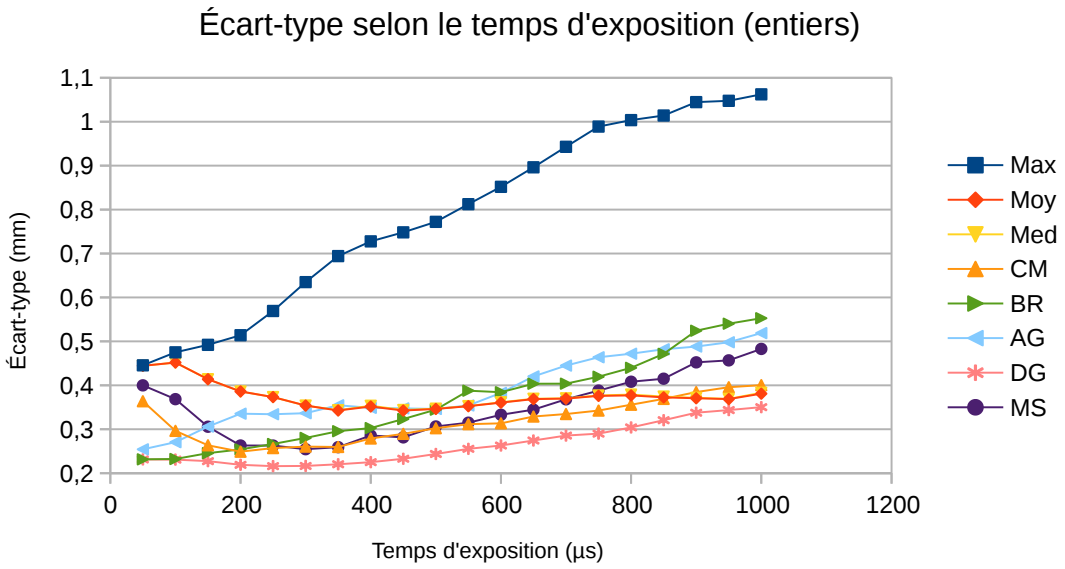


FIGURE 2.10 – Écarts-type selon le temps d'exposition pour la version virgule fixe des algorithmes

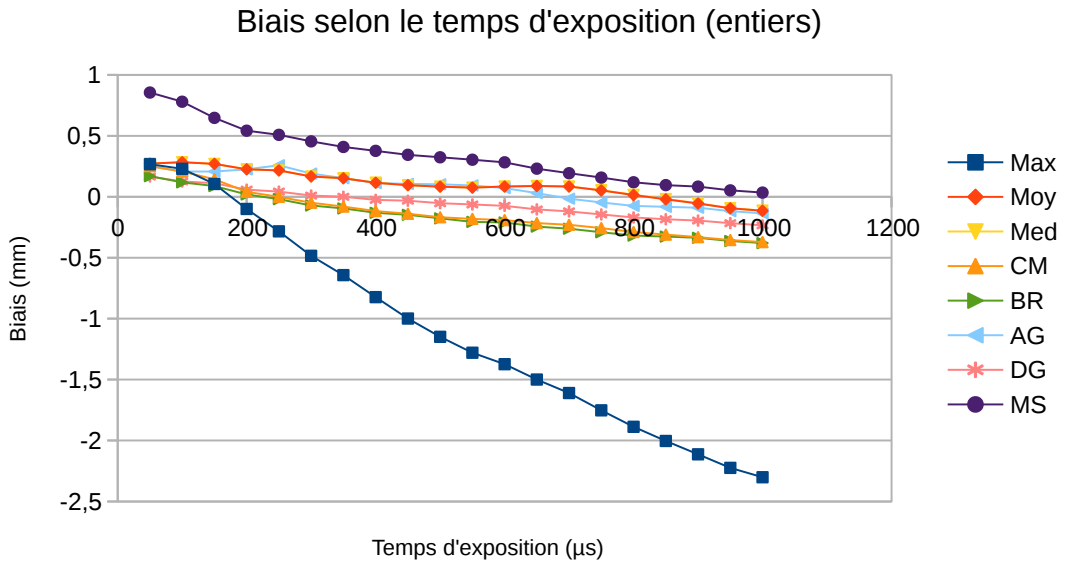


FIGURE 2.11 – Biais selon le temps d'exposition pour la version virgule fixe des algorithmes

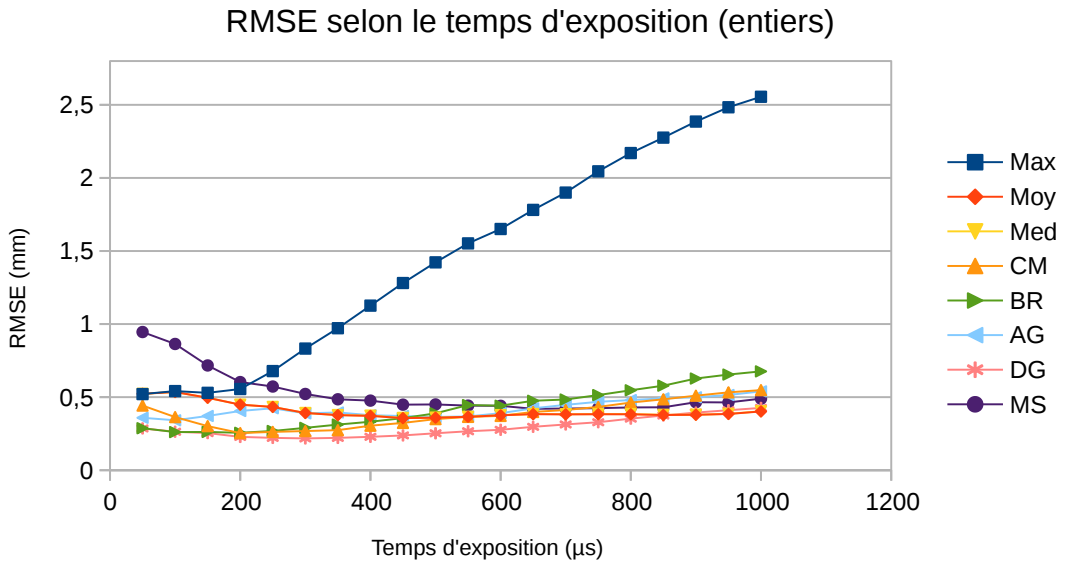


FIGURE 2.12 – Racine de l'erreur quadratique moyenne selon le temps d'exposition pour la version virgule fixe des algorithmes

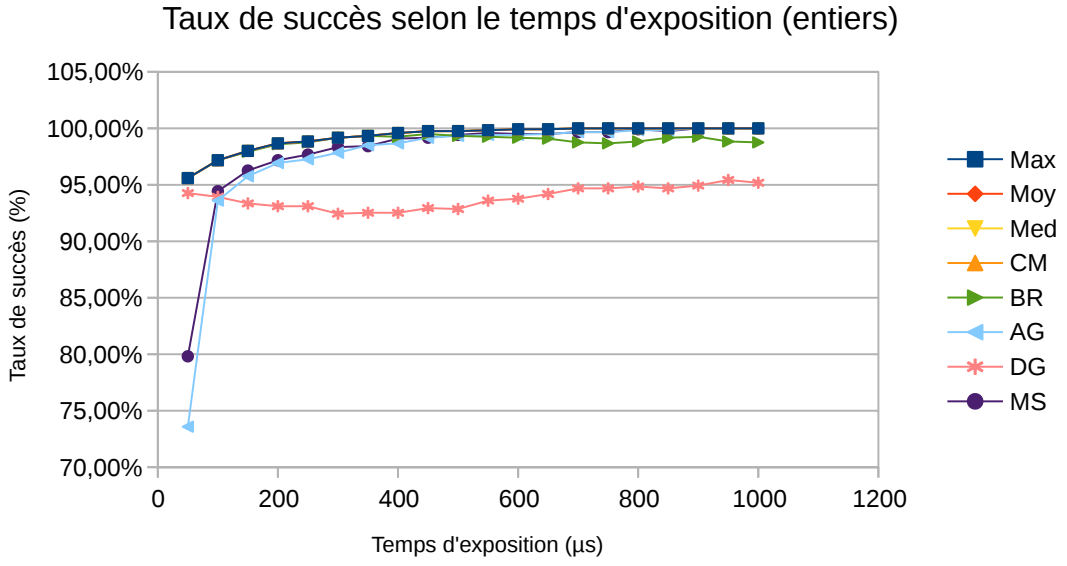


FIGURE 2.13 – Taux de succès selon le temps d'exposition pour la version virgule fixe des algorithmes

des trait lasers de faible intensité. Ceci semble cohérent pour les algorithmes *DG*, *BR* et *AG* au vu des hypothèses nécessaires à l'application de ces algorithmes (trait laser de profil gaussien). À l'inverse, les algorithmes *Moy*, *Med*, *CM* et *MS*, sont moins précis pour des traits lasers de faible intensité. Ces algorithmes là présentent une plage de temps d'exposition dans laquelle ils ont un écart-type très faible (entre 100 et 500 μs dans notre essai). Ceci paraît cohérent avec ce qui a été présenté en partie 2.3.

Pour les temps d'exposition les plus faibles (inférieurs à 300 μs), c'est-à-dire lorsque le trait laser n'est pas ou faiblement saturé, et donc soumis au bruit de Speckle, les algorithmes les plus précis sont ceux basés sur le principe du filtrage FIR (*DG* et *BR*) et l'approximation gaussienne (*AG*). Dans la zone de temps d'expositions moyens (entre 200 et 600 μs) les algorithmes les plus précis sont le *DG*, le centre de masse (*CM*) et les moments statistiques (*MS*). Dans la zone de temps d'exposition élevés (>600 μs), les algorithmes les plus précis sont le *DG*, le max (*Moy* et *Med*) et le centre de masse (*CM*).

Pour le biais, à part les algorithmes *Max* et *MS*, tous les algorithmes ont un biais compris entre -0.5 mm et +0.5 mm. Lorsque le temps d'exposition augmente, donc lorsque l'on sature le trait laser, le biais à tendance à augmenter. Les algorithmes ayant le biais le plus important sont les algorithmes *Max* et *MS*. Les algorithmes ayant le biais le plus faible sont le *DG*, l'approximation gaussienne (*AG*) et la moyenne (*Moy*). Les biais sont plus faibles pour des temps d'exposition moyens (entre 200 μs et 600 μs). Comme pour l'écart-type, des zones où le biais est très faible peuvent être distingués sur la figure. Cela est probablement dû à la calibration. Un biais important n'est cependant pas un problème car il peut être corrigé par la calibration.

Concernant la racine de l'erreur quadratique moyenne, à part les algorithmes *Max* et *MS*, tous les algorithmes ont une précision comprise entre 0.7 et 0.2 mm.

Le taux de succès des algorithmes est relativement faible lorsque les temps d'exposition sont faible, et donc lorsque le trait laser est peu intense et soumis au bruit. Pour les temps d'exposition faibles, les algorithmes *AG* et *MS* sont ceux ayant les taux de succès les plus faibles. Par la suite, plus le trait laser est intense, plus il est facile à détecter. Le seul algorithme présentant un faible taux de succès en dehors de la zone où le laser est peu intense est le *DG*.

Les résultats pour la version flottante double précision des algorithmes sont visibles sur les figures 2.14 à 2.17.

Il y a peu de différences entre les versions flottantes simple précision et flottantes double précision des algorithmes. Nous nous concentrons donc sur la présentation des résultats pour les versions flottantes double précision.

Entre la version virgule fixe et la version virgule flottante, les principales différences sont au niveau des algorithmes *AG*, *CM* et *MS*. L'algorithme *AG* perd 0.1 mm de précision lors de son passage à la virgule flottante pour les temps d'exposition les plus élevés. L'algorithme *CM* pers 0.05 mm de précision sur pratiquement tous les temps d'expositions. L'algorithme *MS* a perdu pratiquement 0.1 mm de précision sur la plage où il était le plus précis.

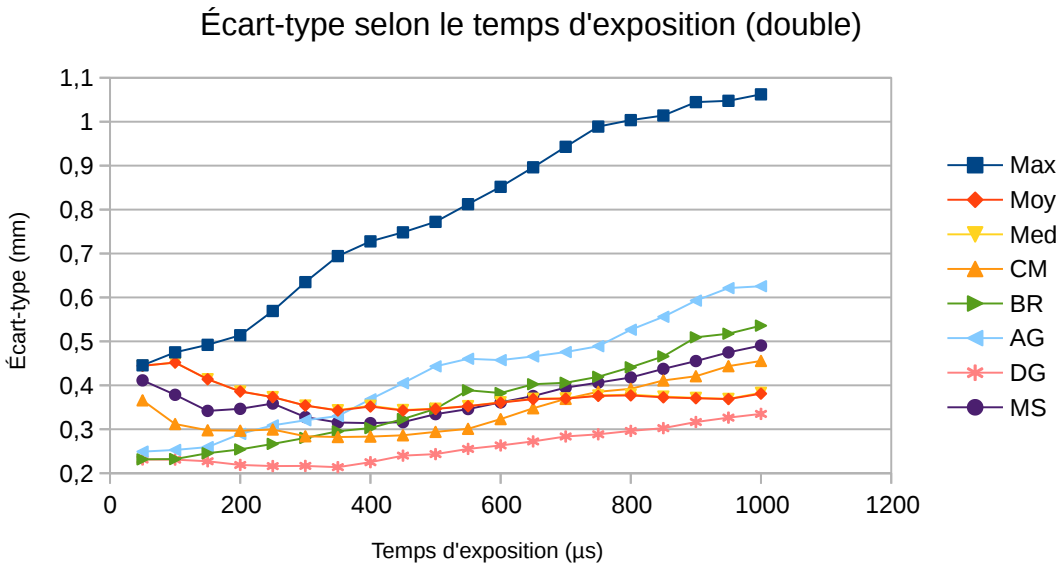


FIGURE 2.14 – Écarts-type selon le temps d'exposition pour la version flottante double précision des algorithmes

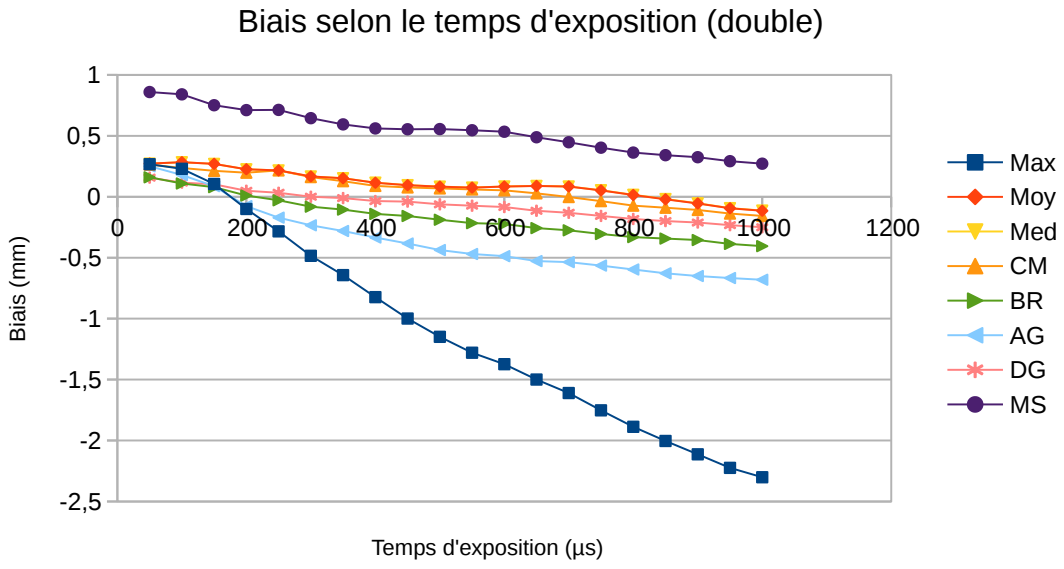


FIGURE 2.15 – Biais selon le temps d'exposition pour la version flottante double précision des algorithmes

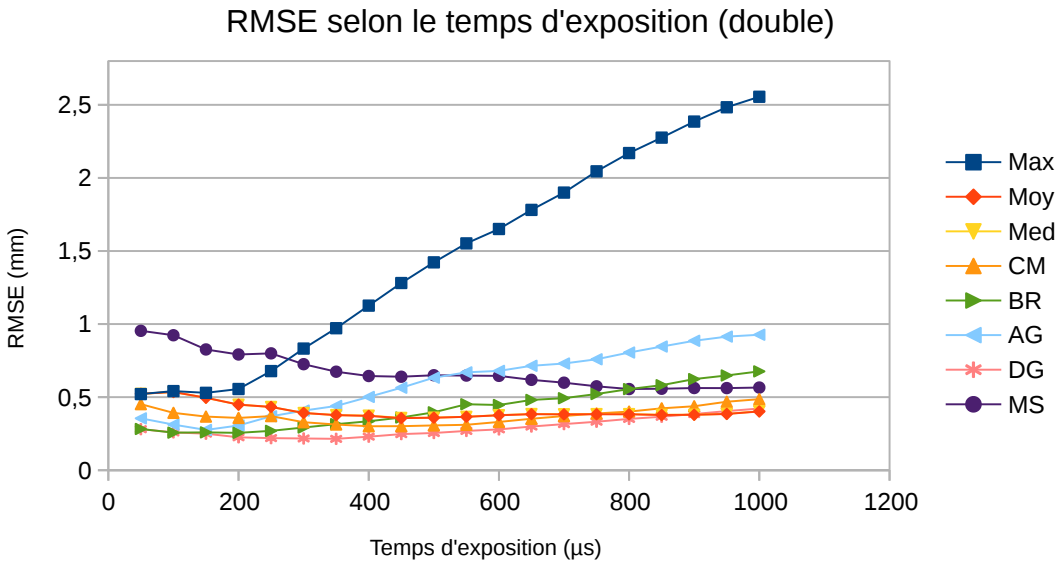


FIGURE 2.16 – Racine de l'erreur quadratique moyenne selon le temps d'exposition pour la version flottante double précision des algorithmes

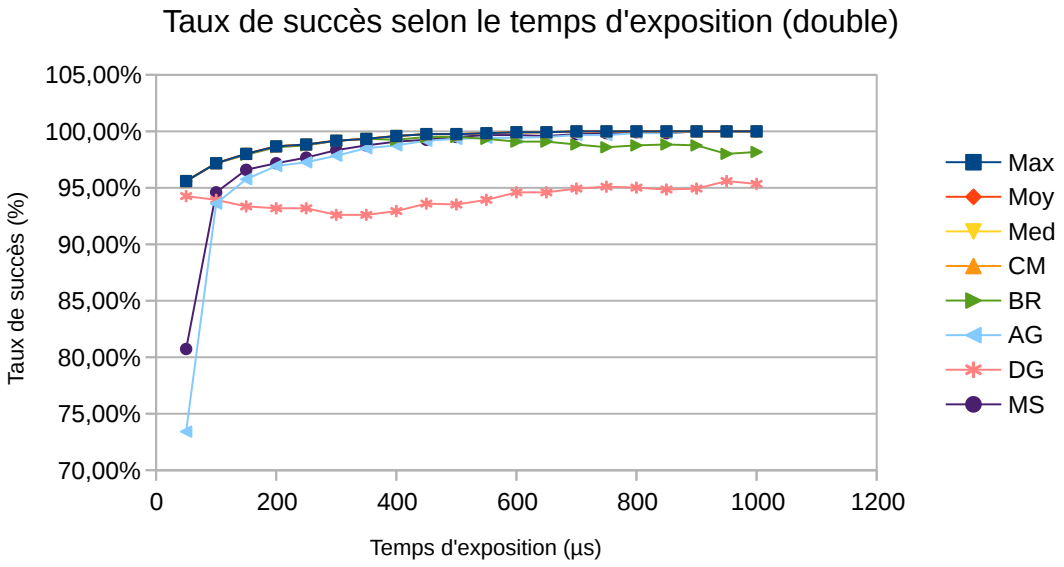


FIGURE 2.17 – Taux de succès selon le temps d'exposition pour la version flottante double précision des algorithmes

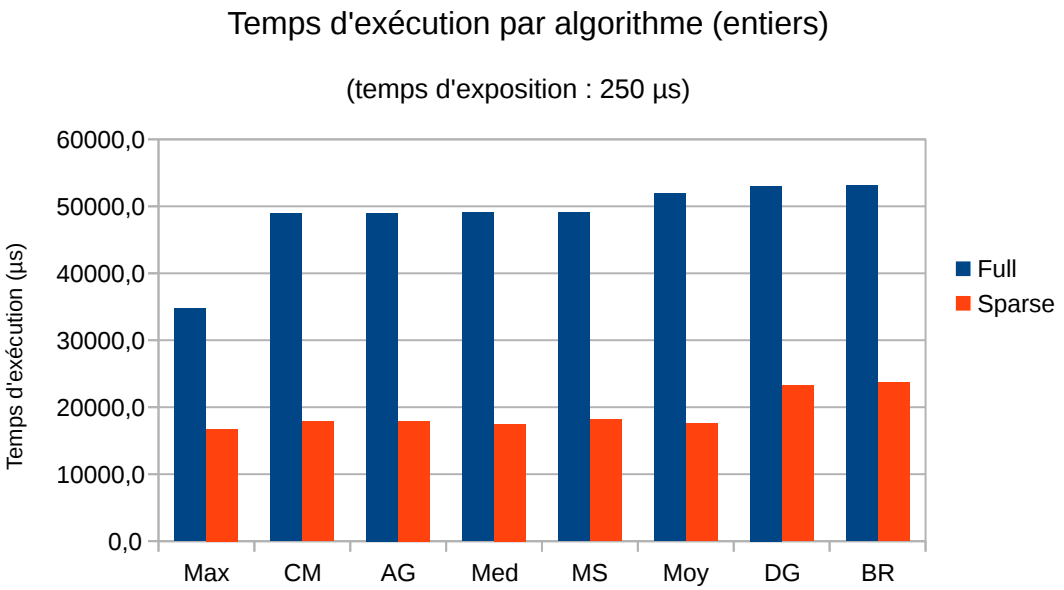


FIGURE 2.18 – Temps d'exécution en μs sur CPU pour la version entière des algorithmes

Au niveau du biais, les algorithmes *AG* et *MS* ont des biais plus importants dans leur version flottante pour les temps d'exposition élevés. L'algorithme *CM* présente à l'inverse un biais plus faible.

Le RMSE des algorithmes *AG* et *MS* en version double a globalement augmenté. Pour l'algorithme *CM*, le RMSE a légèrement augmenté dans la plage où il était le plus précis (entre 100 et 500 μs), mais a légèrement diminué dans la plage où il était moins précis ($>$ 500 μs). Les taux de succès sont inchangés. Seul l'algorithme *BR* voit sont taux de succès légèrement diminuer pour les temps d'expositions les plus élevés.

2.4.3 Temps d'exécution

Les temps d'exécution des différents algorithmes ont été mesurés sur la plateforme logicielle de test. Les algorithmes ont été exécutés 35 fois et la moyenne des temps d'exécution mesurés a été retenue. La figure 2.18 montre les résultats obtenus pour la version entière des algorithmes pour une image ayant un temps d'exposition de 250 μs.

La figure montre des temps d'exécution plus faible pour les fonctions *sparse* (c'est-à-dire celles où l'on ne garde que les données supérieures à un seuil en mémoire) par rapport aux fonctions *full* (celles où l'on traite tous les pixels). L'écart est d'environ de 29,6 ms en faveur des fonctions *sparse*. Sur CPU, les algorithmes en version *full* peuvent être classés dans l'ordre suivant : *Max*, *CM*, *AG*, *Med*, *MS*, *Moy*, *DG* et le *BR*. En version *sparse*, les algorithmes peuvent être classés dans l'ordre suivant : *Max*, *Med*, *Moy*, *CM*, *AG*, *MS*, *DG* et le *BR*. L'algorithme *Max*

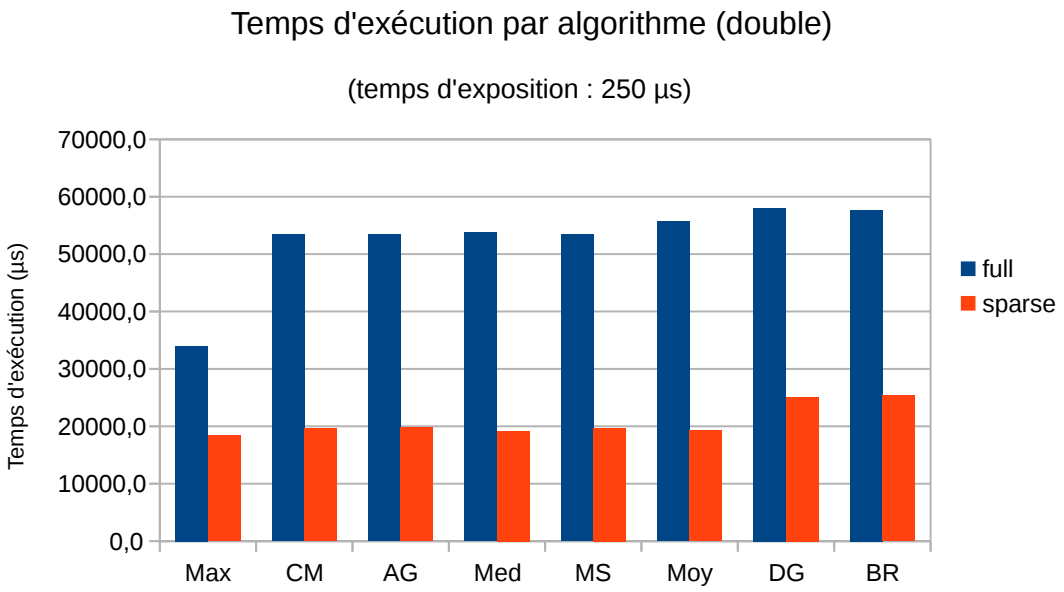


FIGURE 2.19 – Temps d'exécution en μs sur CPU pour la version flottante double précision des algorithmes

est toujours le plus rapide. Les algorithmes *DG* et *BR* sont toujours les plus lents. Les algorithmes *Med*, *AG*, *CM*, *MS* ont des temps d'exécution très proches les uns des autres. L'écart relatif entre le plus rapide et le plus lent de ces algorithmes est inférieur à 4%. Le format *full* ou *sparse* a une influence importante sur le temps d'exécution de l'algorithme *Moy*, qui passe de la 6e à la 3e position.

Les temps d'exécution sur CPU pour la version flottante double précision des algorithmes est visible sur la figure 2.19. Les tendances sont les mêmes pour tous les algorithmes. Cependant, en dehors de l'algorithme *Max*, le temps d'exécution des algorithmes est 9% plus important.

2.4.4 Consommation mémoire

La consommation mémoire de chacun de ces algorithmes sur CPU a été estimée en tenant compte des déclarations de tableaux de données. Pour l'estimation, nous avons noté M la largeur de l'image, N la hauteur de l'image, L le nombre de pixels sur lesquels sont effectués les calculs, L' le nombre de pixels dans la colonne après seuillage, et W la largeur des filtres.

Dans un premier temps, les algorithmes ont été décomposés en des macros-fonctions élémentaires. La taille des tableaux de données nécessaires entre chaque fonctions a été estimée. Les décompositions en fonctions et les estimations de la taille des tableaux de données entre les fonctions peuvent être visibles sur les figures 2.20 et 2.21.

Dans la version *full* des algorithmes, il est nécessaire d'avoir avant l'opération un tableau de taille $M \times N$ stockant la totalité de l'image acquise. Après l'opération,

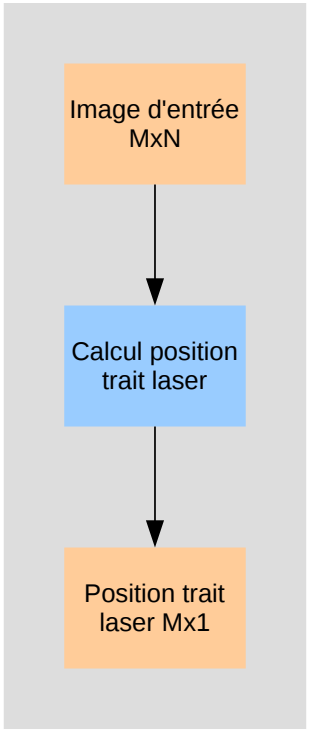


FIGURE 2.20 – Utilisation d'espace mémoire entre les fonctions. Les tableaux sont sur fond orange.

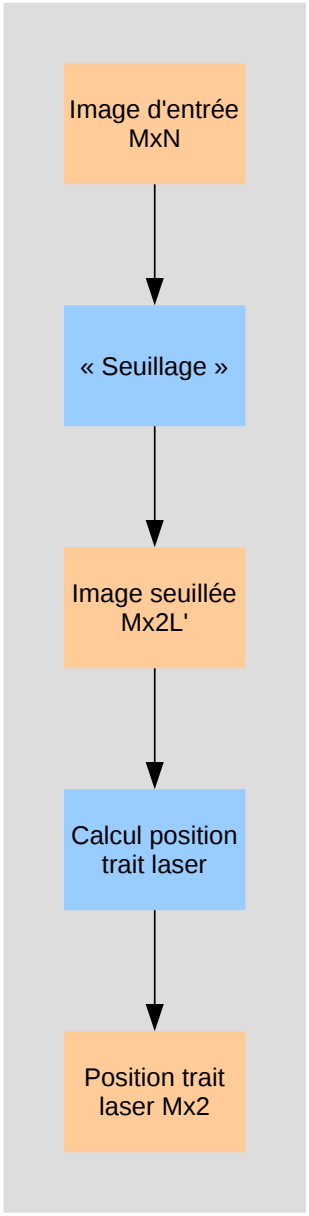


FIGURE 2.21 – Utilisation d'espace mémoire entre les fonctions pour les algorithmes développés en version *sparse*. Les tableaux sont sur fond orange.

TABLE 2.5 – Estimation de la consommation mémoire, en octets, pour chaque algorithme dans leur version *full*

Algorithme	Version entière	Version flottante	Version double
Max	$2.M.N + 4.M$	$4.M.N + 4.M$	$8.M.N + 8.M$
Med, *Moy*	$2.M.N + 4.M + 4.L$	$4.M.N + 4.M + 4.L$	$8.M.N + 8.M + 4.L$
CM, *MS*	$2.M.N + 8.M + 4.L$	$4.M.N + 8.M + 4.L$	$8.M.N + 16.M + 4.L$
AG	$2.M.N + 8.M + 4.L + 9 \times 1024$	$4.M.N + 8.M + 4.L$	$8.M.N + 16.M + 4.L$
BR, *DG*	$2.M.N + 8.M + 4.L + 8.(W + 1)$	$4.M.N + 8.M + 8.(W + 1) + 4.L$	$8.M.N + 16.M + 16.(W + 1) + 4.L$

il est nécessaire d'avoir un tableau de taille $M \times 1$ afin d'y stocker les résultats de la détection de trait laser. La taille de ces tableaux est à multiplier par le nombre de bits ou d'octets des données qu'ils contiennent. La taille des données dépend du format choisi : entiers virgule fixe, flottants simple précision ou flottants double précision.

Dans la version *sparse* des algorithmes, l'étape de calcul de la position du trait laser est précédée par une étape où l'on retire de l'image les pixels dont l'intensité est en-dessous d'un seuil défini. Dans la version CPU, en plus de l'intensité des pixels, la position du pixel dans la colonne est conservée. Il faut donc ajouter un tableau de taille $M \times 2.L$

Si l'on décompose maintenant chacune des "macros"-fonctions en sous-fonctions, nous pouvons estimer la consommation mémoire de chacun des algorithmes. Les format flottants simple et double précision ont été considérés.

Au final, ces estimations mémoire, en octets, sont présentés dans le tableau 2.5 pour la version *full* des algorithmes et dans le tableau 2.6 pour la version *sparse*. La consommation mémoire minimale est de $(M.N + M) \times taille\_de\_la\_variable$ pour la version *full* et de $(M.L' + M) \times taille\_de\_la\_variable$ pour la version *sparse*. La complexité mémoire de ces algorithmes est linéaire par rapport aux variables M, L et W, et quadratique suivant la taille de l'image d'entrée $M.N$ ou $M.L'$.

Les algorithmes consommant le moins de mémoire sont les algorithmes basés sur la recherche du maximum : *Max*, *Med* et *Moy*. La consommation mémoire des algorithmes *Med* et *Moy* est linéaire et dépend des valeurs de L qui est le nombre de valeurs supérieures au seuil de détection, ou encore le nombre de pixels autour du pic d'intensité du laser sur lesquelles les calculs sont effectués. Les algorithmes *CM*, *AG* et *MS* ont une complexité mémoire linéaire dépendant des variables M et L. Cependant, ces algorithmes demandent beaucoup de variables intermédiaires pour calculer la position du trait laser. Les algorithmes basés sur le filtrage FIR, *BR*

TABLE 2.6 – Estimation de la consommation mémoire, en octets, pour chaque algorithme dans leur version *sparse*

Algorithme	Version entière	Version flottante	Version double
Max	$6.M.L' + 6.M$	$8.M.L' + 8.M$	$12.M.L' + 12.M$
Med, *Moy*	$6.M.L' + 6.M + 4.L$	$8.M.L' + 8.M + 4.L$	$12.M.L' + 12.M + 4.L$
CM, *MS*	$6.M.L' + 12.M + 4.L$	$8.M.L' + 16.M + 4.L$	$12.M.L' + 24.M + 4.L$
AG	$6.M.L' + 12.M + 4.L + 9 \times 1024$	$8.M.L' + 16.M + 4.L$	$12.M.L' + 24.M + 4.L$
BR, *DG*	$6.M.L' + 12.M + 4.L + 8.(W + 1)$	$8.M.L' + 16.M + 8.(W + 1) + 4.L$	$12.M.L' + 24.M + 16.(W + 1) + 4.L$

et *DG* nécessitent un peu plus de mémoire. En effet, ces algorithmes nécessitent de stocker les coefficients des filtres en mémoire. C'est pourquoi ils sont dépendants de la largeur W des filtres, en plus de la largeur d'image M et du nombre de pixels sur lesquels sont effectués les calculs L.

Pour l'algorithme *AG*, dans la version entière, cet algorithme utilise des tables pour le calcul du logarithme. Ceci explique le fait qu'il utilise un tableau supplémentaire dans cette version.

2.4.5 Conclusion

Dans cette section, plusieurs caractéristiques des algorithmes ont été étudiées. Ces caractéristiques étaient le nombre d'opérations, la complexité algorithmique, la complexité spatiale, la précision et le temps d'exécution sur CPU. Suivant la métrique utilisée, les algorithmes peuvent être classés différemment. Concernant le nombre d'opérations, les algorithmes les plus intéressants sont *MS*, *Max*, et *Med*. La complexité en $L \times N$ Les algorithmes *BR* et *DG* est un large inconvénient. Concernant la complexité des opérateurs nécessaires, les algorithmes *Max* et *Med* sont ceux qui nécessitent les opérations les moins complexes. A l'inverse, l'algorithme *AG* nécessite des opérations assez complexes (logarithme). En termes de consommation mémoire, les algorithmes *Max*, *Med* et *Moy* sont ceux qui en nécessitent le moins. A l'inverse les algorithmes *BR* et *DG* sont ceux qui nécessitent le plus de mémoire. En termes de temps d'exécution, l'algorithme *Max* est celui qui est le plus rapide, suivi par les algorithmes *CM*, *AG*, *Med*, *MS* et *Moy*. Les algorithmes *DG* et *BR* sont les plus lents.

La précision est l'un des critères de choix d'algorithme les plus importants avec le temps d'exécution. Concernant le format de nombre utilisé, il y a peu de différences entre les versions en virgule flottante et en virgule fixe. Comme le format virgule fixe nécessite moins de ressources ce format peut être utilisée sans perdre

de précision. Comme noté dans la partie 2.4.2, les algorithmes ont des précisions différentes selon le niveau de bruit auquel est soumis le laser. Pour les temps d'exposition faibles (< 200 μs), les algorithmes les plus intéressant sont *BR* et *DG*. Pour les temps d'expositions moyens (> 200 μs et < 500 μs), les algorithmes les plus intéressants sont *DG*, *CM* et *BR*. Pour les temps d'expositions élevés, les algorithmes les plus intéressants sont *DG*, *Moy* et *MS*. Finalement, même si l'algorithme *DG* est le plus précis, son taux de succès 5% plus faible que les autres algorithmes fait qu'il n'est peut-être pas le plus intéressant à utiliser.

2.5 Conclusion

De nombreuses méthodes d'acquisition tridimensionnelles ont été mises en place ces dernières décennies. Ces méthodes ont diverses caractéristiques en termes de vitesses d'acquisitions, de précision et de sensibilité à la lumière notamment. Dans un contexte de contrôle qualité industriel, la méthode la plus adaptée est la triangulation laser.

D'après l'état de l'art, les problématiques de la triangulation laser ont toujours été sa vitesse de traitement, et notamment le nombre de profils traités par seconde, et sa précision malgré la présence de bruit ou de réflexions dues à un produit reflétant particulièrement la lumière. Les problèmes de précision et de temps de calcul de la triangulation laser se jouent au niveau de l'algorithme de détection de trait laser et de la calibration.

Dans cette partie, les principaux algorithmes de détection de trait laser ont été présentés puis comparés suivant des critères tels que la précision, le temps de calcul ou la complexité des opérateurs. En termes de précision, nous retenons l'algorithme *DG*, mais son temps de calcul est l'un des plus élevé. En termes de temps de calcul, l'algorithme le plus rapide est le *Max*, cependant, il s'agit de l'algorithme présentant l'erreur la plus importante. L'algorithme *CM* est un bon compromis vitesse-précision. Il s'agit de l'algorithme qui sera utilisé pour la détection de trait laser dans ce travail.

Chapitre 3 : Architectures génériques pour systèmes de vision sur FPGA

Sommaire

Pour des raisons de pérennité du matériel et de capacité de calcul, nous avons décidé d'implanter notre système de vision sur un FPGA. Les FPGAs présentent des caractéristiques qui en font des composants de choix pour les applications de traitement d'image. La partie 3.1 présente les FPGAs, ainsi que leurs caractéristiques.

La mise en place d'un algorithme sur un FPGA se fait via une architecture matérielle. Le passage de l'algorithme à une architecture matérielle est présenté dans la partie 3.2.

Le développement d'architectures matérielles sur FPGA suit une méthodologie appelée flot de conception. Ce flot de conception est présenté dans la partie 3.3.

Les principaux choix architecturaux effectués pour le système de vision sont présentés partie 3.4

Enfin, la partie 3.5 conclura ce chapitre.

3.1 Présentation des FPGAs

Un FPGA (*Field Programmable Gate Array*) est un composant électronique qui peut être reconfiguré après fabrication. La reconfiguration se fait en modifiant les connexions entre composants et en modifiant la configuration des portes logiques et registres du composant. Un FPGA peut être synchrone à une horloge ou multi-domaine. Ils sont utilisés pour des applications complexes nécessitant d'importantes performances temporelles telles que le traitement du signal, le

traitement d'image, ou les asservissements. Les applications les plus récentes des FPGAs peuvent être trouvées dans [HPM17].

Les principaux fournisseurs de FPGA sont : Xilinx [xil], Altera [altb] (maintenant Intel FPGA), Lattice Semiconductor [lat], et Microsemi [mic]. La suite de cette introduction présente principalement les FPGAs du fournisseur Xilinx étant donné que les travaux présentés dans ce manuscrit ont été effectués sur les FPGAs de ce fournisseur. Cependant, les structures sont similaires pour les autres FPGAs du marché.

Les FPGAs du fournisseur Xilinx sont composés de blocs logiques configurables (CLBs) et de blocs d'entrées-sorties (IOBs) interconnectés par des canaux de routage. Les CLBs sont composés de lookup tables (LUTs) qui permettent d'effectuer les opérations logiques, et de registres qui permettent de stocker les résultats et de se synchroniser avec l'horloge ou les horloges du système. Il est possible d'assembler plusieurs registres pour créer de la RAM distribuée ou des registres à décalage. En plus de ces principaux blocs, les FPGAs contiennent souvent des blocs mémoire internes de type RAM (BRAMs), des blocs DSPs (Digital Signal Processor) contenant multiplicateurs et accumulateurs permettant de créer des fonctions de traitement du signal jusqu'à 500 MHz [ALTc], des boucles à verrouillage de phase (PLLs) permettant de créer des horloges ou encore des blocs émetteurs-récepteurs [XIL16a] pour optimiser les IPs de communication. Ces différents blocs matériels peuvent être assemblés de façon à effectuer du traitement de données en parallèle et/ou en pipeline. Cela constitue l'un des avantages des FPGAs par rapport aux CPUs.

Les FPGAs étant des composants standards reconfigurables, ils sont pérennes sur le marché des composants électroniques. C'est à dire qu'un modèle de FPGA est disponible plusieurs dizaines d'années sur le marché contrairement aux GPUs par exemple. De plus, les langages de descriptions matériel tels que le VHDL ou le Verilog sont des standards qui ne dépendent pas du FPGA utilisé. Ainsi, un opérateur développé pour une gamme de FPGAs peut être porté vers une autre gamme de FPGAs avec pas ou peu de modifications.

Les FPGAs ont également l'avantage d'avoir une consommation électrique faible par rapport aux CPUs et GPUs. En effet, chaque nouvelle gamme de FPGA consomme moins que la précédente. Dans les gammes de FPGA actuelles, le principal challenge est la réduction de la consommation statique. Elle est actuellement entre 300 et 400 mW pour un Spartan 7 ou un Cyclone V, alors que pour un Spartan 6 ou un Cyclone IV, la consommation statique est d'environ 600 mW [Xil17]. A l'inverse, les CPUs Intel les plus économes en énergie consomment au minimum 3 Watts au repos [har17]. Cependant, la plupart des CPUs ARM consomment très peu d'énergie : entre 25 et 606 mW pour un Cortex A8 [Tom09]. Pour les GPUs, la consommation au repos (écran éteint) varie de 3 à 30 Watts selon les modèles [Les16].

Néanmoins, les FPGAs ont également quelques inconvénients qui sont des temps de développement plus longs que sur CPU et la difficulté du débug. De plus, ils sont peu performants sur des opérations séquentielles telles que du contrôle ou

des lectures et écritures dans une RAM externe.

Les outils de synthèses FPGA fonctionnent habituellement au niveau RTL (*Register-Transfer Level*). Il s'agit d'un niveau d'abstraction où le modèle de calcul est représenté à partir de signaux constituant un flux de données subissant des opérations logiques entre des registres. Récemment, des outils de synthèse « haut niveau », permettant de générer des bitstream ou du code niveau RTL à partir de code C/C++ ou Python ont vu le jour pour pallier à cet inconvénient de temps de développement et debug [DZS14] au détriment des performances.

Depuis 2015, les FPGAs gagnent progressivement en popularité. Le nombre de transistors par puce ayant fortement augmenté, il est maintenant possible de créer des systèmes complexes sur FPGA incluant des CPUs (SoCs, NoCs ou MPSoCs). Cette popularité nouvelle des FPGAs est principalement due à son utilisation dans les datacenters de grandes entreprises telles que Amazon [Fre16] ou Microsoft. Les FPGAs sont également un composant de choix pour les réseaux de neurones [OR06], les applications de type deep learning [LTA16] ou machine learning, le traitement d'images ou le traitement du signal.

Enfin, certain FPGAs ont l'avantage d'être partiellement reconfigurable dynamiquement. C'est-à-dire que pour un FPGA déjà configuré, il est possible de reconfigurer une partie seulement du système présent sur le FPGA lors de son fonctionnement [McD08]. Cependant, tous les FPGAs ne possèdent pas cette caractéristique. Chez Xilinx, la reconfiguration partielle est supportée pour tous les FPGAs de la famille Virtex (4, 5, 6 et 7) et de la famille Kintex 7. Quelques FPGAs des familles Artix 7 et Zynq 7000 supportent également cette fonctionnalité.

3.2 Conception d'architectures matérielles

La conception d'architectures matérielles se fait à partir d'un algorithme. Un algorithme est décrit sous forme de tâches liées par des flux de données, de contrôle ou mixtes. Cet algorithme peut être représenté avec un modèle de calcul. Il existe principalement deux modèles de calculs : les modèles flux de données et les modèles flux de contrôle. Dans les modèles flux de données, les résultats générés après l'application d'une première instruction sont transmis directement à l'instruction suivante afin d'y être traités. Ils sont représentés sous forme de graphes flot de données ou « data flow graphs ». Dans les modèles flux de contrôle, les résultats sont stockés et partagés avec toutes les instructions. C'est le modèle de calcul utilisé par les compilateurs.

L'architecture matérielle est réalisée en assemblant des modules plus ou moins complexes, appelés *Intellectual Property* ou IP. Ces modules ou IPs consistent en des opérateurs de calcul, des opérateurs de contrôle, des blocs mémoire ou un assemblage de plusieurs d'entre eux. Chaque tâche de l'algorithme peut donc être réalisée par une ou plusieurs IPs de calcul ou de contrôle. En entrée et sortie de chaque tâche, des données doivent être mémorisées. Pour cela, des IPs de mémorisation peuvent être utilisées. Pour le transfert des données entre chaque tâche,

des protocoles de communication et des interfaces entre les IPs peuvent être utilisés. Ces protocoles de communication peuvent être entre IPs ou vers l'extérieur. Le contrôle peut être distribué ou centralisé. Un contrôle est dit distribué lorsque chaque opération s'effectue indépendamment les unes des autres. Ainsi, aucune autre IP n'intervient dans l'exécution de ces tâches. À l'inverse, un contrôle est dit centralisé lorsqu'une IP contrôle la majorité des autres IPs. Les IPs de contrôle centralisé les plus populaires sont les processeurs.

La conception d'architectures matérielles peut se faire avec des langages de description matériel également connus sous le nom de HDL (*Hardware Description Langage*). Les deux principaux langages de description matériel existants sont le VHDL et le Verilog. Elle peut également être réalisée avec des langages de haut niveau en passant par des outils de synthèse haut niveau (HLS). S'il y a présence de processeurs, des langages de développement logiciel tels que le C peuvent également être utilisés. La création d'architectures matérielles respecte une méthode appelée flot de conception.

3.3 Flot de conception FPGA

Le flot de conception est l'ensemble des étapes permettant de développer des opérateurs et systèmes sur FPGA. Il est représenté sur la figure 3.1. Cette partie présente les différentes étapes du flot de conception sur FPGA.

3.3.1 Contraintes du système

Cette première étape permet de décrire les contraintes que doit respecter le système à mettre en œuvre. Elle consiste à préciser les FPGAs supportés, la fonctionnalité du système, les fréquences d'horloges nécessaires à son bon fonctionnement (contraintes de timing), ainsi que les contraintes en termes de consommation. Si certaines IP du système doivent respecter une norme ou un standard, il faut préciser quelle norme ou standard, ainsi que sa version, et les options du standard qui seront supportées dans la version finale.

3.3.2 Spécifications du système

Cette étape consiste à [KB07] :

- Déterminer un modèle comportemental pour la fonctionnalité à mettre en œuvre, notamment, choisir un algorithme qui répond aux contraintes du système souhaité (par exemple la résistance au bruit, la précision du résultat, le temps de calcul...). Pour cela, il est possible de s'aider de langages de vérification matériel tels que VSPEC, SystemC ou SystemVerilog.
- Déterminer le partitionnement hardware/software s'il y a présence d'un processeur, ainsi que les standards d'interconnexion à mettre en place entre IPs et entre IPs et processeurs.

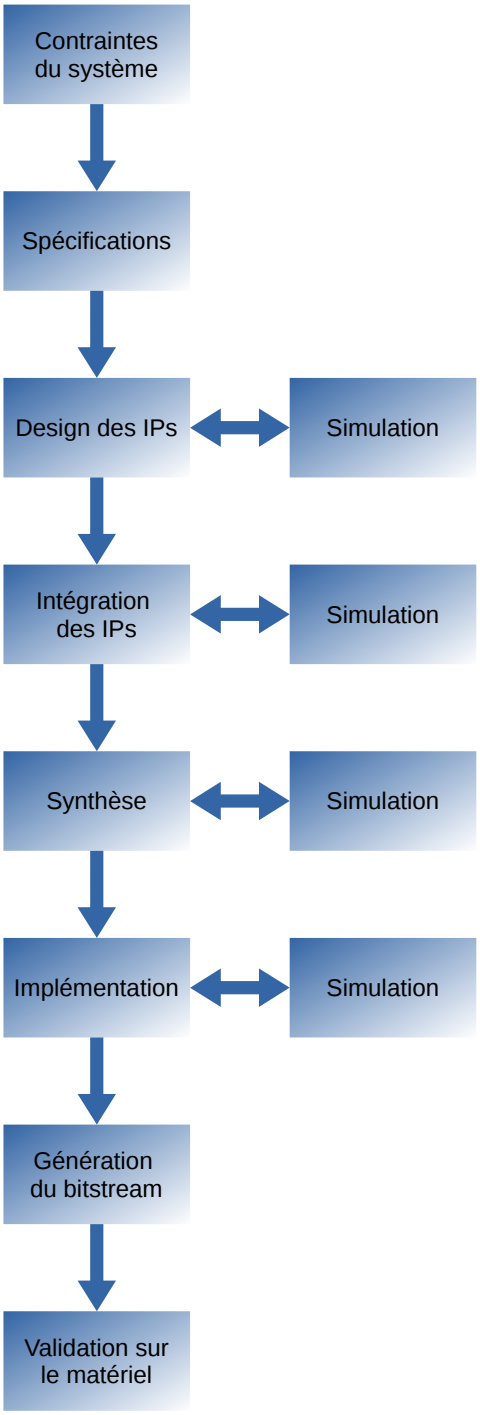


FIGURE 3.1 – Flot de conception FPGA

- Déterminer une architecture matérielle en sélectionnant les structures matérielles (blocs logiques, mémoires, bus de communication...) qui vont être mises en place, en s'assurant que le modèle final répond aux contraintes définies précédemment.

3.3.3 Design des IPs

La première étape consiste à définir les caractéristiques de l'IP telles que sa fonctionnalité, ses paramètres génériques, ses registres accessibles par soft, ses entrées/sorties et ses contraintes en termes de fréquence d'horloge, de place utilisée sur le FPGA et de consommation électrique.

Une fois l'opérateur et ses caractéristiques définis, il est possible, au choix :

- de les développer manuellement avec un langage HDL (VHDL ou Verilog) ou un langage haut niveau (type C/C++ ou SystemC [sys]),
- de les acheter auprès de fournisseurs d'IPs,
- de les télécharger sur des sites dédiés aux IPs open source,
- d'utiliser les outils du fournisseur FPGA qui permettent de générer des IPS.

Il est courant de développer manuellement les IPs personnalisées ayant une forte valeur ajoutée sur le produit final et d'utiliser des IPs achetées ou générées automatiquement pour des opérations standards et complexes telles que les bus, les protocoles de communication (ethernet, USB...), ou les processeurs.

Si besoin, des drivers C sont également développés afin de s'interfacer avec un processeur, soit après le développement matériel de l'IP, soit parallèlement au développement matériel de l'IP avec un modèle comportemental précis du matériel (cosimulation).

Un modèle de simulation fonctionnelle est également créé pour simuler l'IP développée. Les IPs sont simulées avec un simulateur fourni soit par le fournisseur du FPGA (Xilinx, Altera, Lattice,...), soit par des tiers (Synopsys, Mentor Graphics,...).

Si les IPs sont interfacées avec des processeurs, il faut implémenter les drivers de l'IP en C.

3.3.4 Intégration des IPs

Après l'étape de design des IPs, nous nous retrouvons avec des IPs de type :

- Opérateur de calcul ou contrôle,
- Communication,
- Mémoire.

L'étape d'intégration consiste à interconnecter ces IPs afin de créer un système complet sur FPGA. Ce système peut exploiter un parallélisme de flux (pipeline), et/ou un parallélisme de données.

Pour que l'intégration soit le plus simple possible, il est intéressant d'avoir des protocoles de transfert de données prédéfinis entre les IPs. Actuellement, le standard le plus populaire est l'AMBA AXI qui est supporté par les FPGAs Xilinx

TABLE 3.1 – Standards d'interconnexion IPs et IPs/CPUs

Standard	Point à point	Bus	Autres
AMBA	AXI Streaming	AXI MM, AXI-Lite, AHB, APB	
Avalon	Avalon-ST	Avalon-MM	Avalon Conduit, Avalon-TC
Wishbone	Point-to-point, Dataflow	Shared bus, Crossbar	

[Xil11] et Altera [Cor13]. Cependant, Altera (maintenant Intel FPGA) a également son propre standard qui est l'Avalon. Il existe également un standard open source qui est le Wishbone [Ope10].

Le choix de ces standards dépendent principalement du processeur utilisé. Néanmoins, la plupart de ces standards ont des points communs notamment au niveau du type d'interconnexion et des signaux nécessaires. Parmi les différents types d'interconnexions, il y a :

- Les interconnexions points à points de type flot de données, notamment utilisées pour faire du pipeline.
- Les interconnexions de type bus qui permettent de relier un ou plusieurs maîtres à plusieurs esclaves. Ces bus peuvent être lents ou rapides. [MS06] en ont fait une review.
- Les interconnexion de type réseau également appelés NoCs. Ce type d'interconnexion permet de relier ensemble un nombre important d'IPs communicantes.

Le tableau 3.1 résume les types d'interconnexions selon le standard utilisé.

S'il y a présence d'un processeur, le code source du CPU est développé. Ce développement est effectué en respectant le partitionnement hardware/software défini à l'étape de spécification. Le développement du code source du ou des CPUs peut être fait, après les étapes d'intégration, de synthèse, de placement-routage, et de génération du bitstream. C'est-à-dire après la création de la plate-forme matérielle. Mais ce développement de code CPU peut également être effectué parallèlement au développement de la plate-forme matériel avec un modèle comportemental précis du matériel (cosimulation).

Après intégration, il est possible de simuler le système complet ou une partie du système en utilisant des modèles de bus fonctionnels (Bus Functionnal Model ou BFM) permettant de simuler le transfert de données dans un bus.

3.3.5 Synthèse

L'étape de synthèse convertit l'architecture matérielle développée (code VHDL ou Verilog) en une netlist qui est un format représentant un schéma électrique.

Cette étape peut révéler quelques erreurs non décelées lors de la simulation fonctionnelle. La synthèse peut être effectuée avec les outils du fournisseur FPGA ou par des logiciels tiers. Il est possible de réaliser une simulation du système après synthèse. Cette simulation basée sur la netlist est plus précise que la simulation fonctionnelle effectuée avant synthèse.

3.3.6 Implémentation

Lors de l'étape d'implémentation, la netlist est rattachée à un FPGA précis. Les composants de la netlist sont placés sur les structures internes du FPGA telles que les BRAMs, les LUTs, les IOs, et les registres. Puis, ces ressources sont routées en respectant les contraintes du FPGA (placement-routage). Il est possible d'ajouter des contraintes de placement et de timing lors de la réalisation de cette étape. Après le placement-routage, un fichier d'analyse de timing est disponible. Ce fichier permet de faire une simulation intégrant les temps de propagation. Il permet de contrôler le respect des contraintes de timing et d'analyser les causes d'un non-respect des contraintes de timing.

3.3.7 Génération du bitstream

La dernière étape du flot de conception est la génération du bitstream. Le bitstream est le fichier binaire permettant de configurer le FPGA. Il est généré à partir du résultat de l'étape de placement-routage et du fichier de contraintes utilisateurs qui associe les signaux aux PINs du FPGA avec leurs caractéristiques électroniques. Une fois le bitstream obtenu il est possible de configurer le FPGA avec le bitstream et le code source du ou des CPUs (au format .elf) afin de tester et de valider le système créé. Il est également possible de générer un fichier binaire à flasher ou à sauver en EEPROM afin de ne pas avoir à reconfigurer le FPGA après chaque coupure de courant.

En cas de bugs, il est possible d'utiliser les outils de debug mis en place par le fournisseur de FPGA. Pour Xilinx, il s'agit des outils Chipscope (qui est une sorte d'oscilloscope permttant d'observer les signaux internes au FPGA si l'on a pris le temps d'ajouter ce type de composant dans le système) et le debugger proposé par l'outil Xilinx Platform Studio pour debugger le code des CPUs (cela nécessite néanmoins une liaison série entre le PC de développement et le FPGA).

3.4 Système de vision sur FPGA

Le système à mettre en place est constitué d'un capteur image couplé à un FPGA dont la fonctionnalité principale est de traiter en temps réel les images acquises. La première étape de la conception d'un système de vision sur FPGA est de mettre en place l'interface entre le capteur et le FPGA.

Le tableau 3.2 présente des capteurs images dédiés à des applications de type machine vision. Ces capteurs ont un débit pixel élevé qu'il est difficile de traiter

TABLE 3.2 – Capteurs image intéressants pour les applications machine vision

Référence	Résolution	Vitesse d'acquisition	Débit maximal
CMOSIS CMV2000	2048 x 1088	338 images/s	753 Mpixels/s
ON Semi VITA25K	5120 x 5120	53 images/s	1 389 Mpixels/s
ON Semi VITA12K	4096 x 3072	160 images/s	2 013 Mpixels/s
CMOSIS CMV12000	4096 x 3072	300 images/s	3 774 Mpixels/s

en temps réel. Par exemple, si l'on traite 1 pixel par cycle d'horloge, traiter une image de résolution 4096x3072 à 160 images/s revient à traiter un débit de 2 013 Mpixels/s. Ce système nécessiterait donc une fréquence d'horloge d'environ 2 GHz pour effectuer ce traitement. La seconde étape est donc de concevoir des opérateurs de traitement d'image temps réel permettant de traiter ce débit image.

Dans notre contexte, les résultats du traitement sont transmis à un PC industriel via une communication Ethernet. Cette interface doit donc être mise en place.

Certains traitements nécessitent de stocker une image entière. Hors, une image de résolution 4096x3072 au format 10 bits, nécessite 15 Mo de mémoire. Une image de cette taille ne peut être stockée dans les BRAMs de FPGAs low-cost. Ainsi, la majorité des systèmes de vision sur FPGA possède une interface avec RAM externe permettant de stocker des images entières.

Chacune de ces interfaces et fonctions doivent pouvoir être contrôlées. Cette partie sera également étudiée.

3.4.1 Acquisition de l'image

Dans un premier temps, l'interface entre le capteur et le FPGA doit être mise en place. Les capteurs transmettent les images acquises via des interfaces de communication spécifiques. Ces interfaces peuvent être de type : LVDS [Ins08], MIPI [mip], HiSPi [Cor03], analogique ou autre. Des exemples ou description d'interfaces avec des capteurs image sont disponibles dans [Def12], [alt13], [alta] ou [Fen10]. En plus de l'interface permettant le transfert de l'image, les interfaces permettant le contrôle du capteur doivent être ajoutées. Un exemple d'interface de contrôle du capteur est l'interface SPI.

Dans le FPGA, les signaux bruts reçus doivent être débruités, désérialisés et mis en forme de façon à pouvoir être transmis aisément via les interconnexions du système. Pour les capteurs couleur, la conversion au format RGB est souvent effectuée lors de cette étape. Concernant la mise en forme de l'image, il n'y a pour le moment aucun standard commun. Cela signifie que chaque fournisseur d'IP propose son propre standard. Par exemple, Xilinx a mis en place le protocole « Video IP » [Xil16b] composé de 5 signaux qui sont :

- *Video Data*
- *Valid*

- *Ready*
- *Start of frame*
- *End of line*

avec *Video Data* correspondant aux pixels de l'image, *Valid* et *Ready* des signaux permettant de respecter un protocole « handshake », enfin, *Start of frame* un signal indiquant le début d'une image et *End of line*, un signal indiquant une fin de ligne.

Chez Altera, le standard mis en place est Avalon-ST Video [Alt17]. Il est constitué des signaux *startofpacket*, *endofpacket*, *data*, *empty*, *valid* et *ready*. D'autres protocoles vidéo utilisent les signaux *Data*, *HSync* et *VSync* pour synchroniser les signaux. Cependant, les fournisseurs de caméra-FPGA mettent souvent en place leur propre protocole de transfert d'images. Le fait que chaque fournisseur ait son propre standard est une limite importante à la généricité du système de vision.

À partir de ce niveau, deux possibilités existent. La première possibilité consiste à traiter directement les données acquises par le capteur dans un pipeline [Mat13]. La seconde possibilité consiste à préalablement stocker l'image en RAM externe avant d'effectuer des traitements [RGFSDR12].

3.4.2 Traitement d'image sur FPGA

Le principal défi au niveau du système de vision sera de traiter le débit de données visible sur le tableau 3.2. Ce débit important à traiter entraîne des contraintes de performance temporelle assez importantes sur les fonctions de calcul développées. Les fréquences d'horloge nécessaires au traitement d'un pixel par cycle sont beaucoup trop élevée pour la plupart des FPGAs. Il est donc important d'exploiter les possibilités du FPGA en termes de parallélisation. Dans notre contexte, le parallélisme de données et le parallélisme de flux vont être utilisés. Ainsi, les différentes fonctions de traitement d'image seront enchaînées en pipeline avec des niveaux de parallélisme permettant de traiter plusieurs données simultanément. Cela permettra de diminuer la fréquence d'horloge sans réduire le débit pixel.

Si l'on reprend les capteurs présentés dans le tableau 3.2 et que l'on estime la fréquence d'horloge minimale nécessaire selon le degré de parallélisme, on obtient les valeurs présentes dans le tableau 3.3. Ainsi, il est possible de choisir le degré de parallélisme selon la fréquence d'horloge minimale du système.

Choisir de mettre en place un pipeline influence le choix du type de connexion entre les opérateurs. Étant donné que les opérateurs de traitement d'image seront enchaînées, il faut entre chaque opérateur de traitement d'image une interconnexion unidirectionnelle et rapide. Pour cela, les interconnexions point à point de type streaming seront privilégiées. Les interconnexions de type bus et NoCs sont superflus dans ce cas et ne permettent pas une performance temporelle élevée.

3.4.3 Communications avec l'extérieur

Les données acquises et traitées par la caméra-FPGA peuvent au choix, être affichées sur un écran, utilisées pour contrôler des actionneurs ou être envoyées à

TABLE 3.3 – Fréquence d'horloge selon le nombre de pixels en parallèle pour les capteurs de la table 3.2

Parallélisme	CMV2000	CMV12000	VITA12K	VITA25K
1	753 MHz	3.78 GHz	2.01 GHz	1.39 GHz
2	377 MHz	1.89 GHz	1.01 GHz	695 MHz
4	188 MHz	943.7 MHz	503 MHz	347 MHz
8	94 MHz	471.8 MHz	252 MHz	173.7 MHz
16	47 MHz	235.9 MHz	125 MHz	86.8 MHz
32	23,5 MHz	118 MHz	62.9 MHz	43.4 MHz
64	11,8 MHz	59 MHz	31 MHz	21.7 MHz

un autre système (par exemple un PC) afin de subir des traitements supplémentaires. Les données peuvent être envoyées brutes ou compressées. Les auteurs de [HMT10] ont mis en place plusieurs interfaces avec l'extérieur sur leur système. Les images traitées ainsi que les résultats sont affichés sur écran VGA. Les résultats du traitement d'image sont également utilisés pour contrôler un actionneur qui sert à déplacer la caméra. Les auteurs de [ISA+14] ont mis en place une interface avec un émetteur radio pour le transfert de l'image traitée. Ils ont également choisi de compresser l'image avant transfert afin de réduire la quantité de données à transférer. Dans notre contexte, les images résultantes obtenues correspondent à une image d'une ligne dont les intensités représentent la position du trait laser. Ces images d'une ligne doivent être transmises à un PC industriel pour un traitement haut niveau.

L'envoi des résultats nécessitent la mise en place d'IPs de communication au niveau du FPGA. Selon les débits nécessaires, il peut s'agir d'un UART, d'une IP Ethernet, voire d'une IP PCi Express. Dans notre contexte applicatif, après détection du trait laser sur l'image, le débit image est divisé par la hauteur de l'image. Ceci permet d'utiliser des protocoles de transfert d'image plus lent que le débit du capteur. Pour des raisons de coûts, et de formation des techniciens, le protocole choisi dans ce contexte est l'Ethernet.

3.4.4 Gestion des mémoires externes

La majorité des systèmes de vision sur FPGA stockent préalablement l'image acquise par le capteur dans une RAM externe avant de traiter cette image [HMT10, RGFSDR12]. Étant donné que les performances temporelles des systèmes de vision sont limitées par ces accès mémoire, des solutions ont été mises en place pour pallier ce problème. Les auteurs de [Mat13] ont modifié leur algorithme de façon à ne pas nécessiter de mémoires externes. Les images sont donc traitées directement. Les auteurs de [ISA+14] ont utilisé des méthodes de compression - décompression de la taille d'image pour pouvoir utiliser moins de mémoire externe. Une autre solution

est d'utiliser des mémoires plus rapides. Les auteurs de [RGFSDR12] ont utilisé une mémoire de type ZBT qui est plus rapide qu'une RAM standard, pour le stockage temporaire des images.

Dans notre contexte, une mémoire externe est nécessaire afin de tamponner les données avant transmission par l'IP Ethernet. De plus, certaines opérations telles que la soustraction d'images, nécessitent de stocker au moins une image en mémoire. Lorsque les images ont une résolution élevée, il est préférable d'utiliser une mémoire externe, qui a une taille plus importante, afin de garder les blocs RAMs et LUTRAMs pour des opérations plus "critiques". Ainsi, dans l'optique de développements futurs et de la mise en place d'un système de vision générique/adaptable, il est intéressant d'avoir une mémoire externe et rapide (RAM).

L'utilisation d'un bloc mémoire externe au FPGA impose quelques contraintes dans l'architecture du système de vision. Elle impose d'abord l'utilisation d'un contrôleur mémoire qui va effectuer les lectures et les écritures dans la mémoire externe. Afin d'automatiser un peu la transmission des données entre le pipeline et la mémoire externe, ainsi qu'entre la mémoire externe et l'IP Ethernet, des DMA peuvent être utilisés. Étant donné que plusieurs IPs doivent être connectées au bloc mémoire externe, l'interconnexion à mettre en place dans cette partie de l'architecture doit être de type bus. Une interface Memory Map est préconisée dans ce cas-là.

3.4.5 Contrôle

En résumé, en plus d'effectuer les opérations de traitement d'image, l'architecture matérielle créée doit : gérer une communication avec un PC industriel via un protocole Ethernet, gérer des accès (lecture et écriture) en mémoire externe et gérer le contrôle du capteur image (lancement et arrêt de l'acquisition entre autres). Ces tâches doivent être gérées par des IPs de contrôle. Le contrôle mis en place peut être distribué ou centralisé.

La plupart des tâches citées ci-dessus étant séquentielles, elles peuvent être gérées par un contrôle centralisé. Ici, il s'agira d'un CPU.

Le CPU ajouté peut être softcore (Xilinx MicroBlaze, Altera Nios II) ou hardcore (Xilinx Zynq SoC, Altera Cyclone V SoC). Les CPUs softcore sont des IPs à placer dans le FPGA au même titre que les opérateurs de traitement d'image. Ils ont l'avantage d'être flexibles, c'est-à-dire qu'il est possible de les changer s'ils ne répondent pas à nos besoins. Leur principal inconvénient est de consommer des ressources dans le FPGA. Les CPUs hardcore sont présents physiquement auprès du FPGA. Le principal inconvénient est la perte en termes de flexibilité, mais ces CPUs ne consomment pas de ressources dans le FPGA. Dans notre contexte, l'un ou l'autre peut convenir.

Pour la partie traitement d'image, un contrôle distribué a été mis en place. C'est à dire que les opérations de contrôle, par exemple la synchronisation des lignes et colonnes de l'image, se fait à l'intérieur de chaque bloc de traitement d'image sans intervention extérieure. Ainsi, il n'y a pas, ou peu d'interventions du

CPU sur cette partie.

L'ajout d'un CPU permet d'ajouter davantage de flexibilité dans le système. Il peut notamment servir à modifier des paramètres dans le système de vision sans avoir à reconfigurer le FPGA en utilisant des registres accessibles dédiés dans les opérateurs. L'ajout d'un CPU dans le système est donc un choix pertinent dans la mise en place d'un système de vision générique sur FPGA.

3.5 Conclusion

Dans ce chapitre, les FPGAs ainsi que les méthodes de conception d'architectures matérielles ont été présenté. Les FPGAs sont des composants électroniques programmables et massivement parallèles qui permettent d'accélérer les calculs.

Les systèmes de vision sur FPGA doivent respecter un certain nombre de contraintes dans leur mise en place afin d'être performants. L'exploitation des différents types de parallélisme et des différents types de flux (données, contrôle...) est important afin d'atteindre les performances temporelles souhaitées.

Chapitre 4 : Mise en œuvre dans notre contexte

Sommaire

4.1 Système de détection de trait laser

Le système de vision à mettre en œuvre est un système de détection de trait laser basé sur le calcul du centre de masse. Ce système est constitué des éléments suivants :

- Un filtre gaussien ou médian afin de lisser légèrement l'image et réduire le bruit HF.

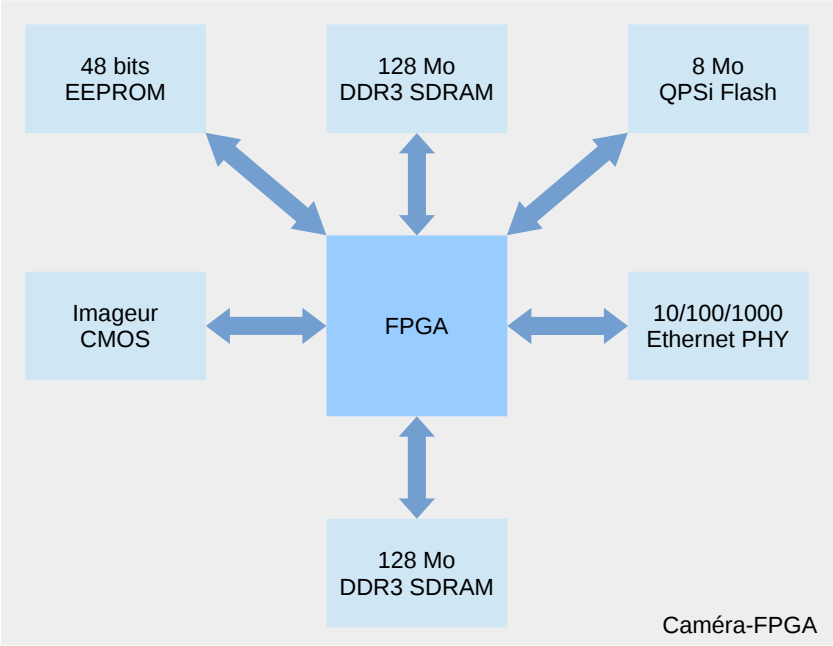


FIGURE 4.1 – Architecture de la caméra-FPGA

- Le module de détection de trait laser.
- Une redynamisation du résultat afin que celui-ci tienne sur 16 bits.

La partie la plus importante du travail de recherche a été accomplie pour la création de l'IP ou module de détection de trait laser. C'est sur cette IP que va se concentrer le reste du manuscrit.

4.2 Caméra utilisée pour le prototypage

Nous avons décidé d'utiliser une caméra-FPGA standard du commerce. Le modèle que nous avons choisi pour le prototypage est « *ouverte* ». Cela signifie qu'il est possible de modifier tous les éléments du système de vision présents sur la caméra.

4.2.1 Présentation de la caméra

La caméra utilisée est une Optomotive Velociraptor Evo [opt]. Son architecture est présentée dans la figure 4.2. Il s'agit d'une caméra contenant un capteur CMOSIS CMV2000, câblé par lignes LVDS à un FPGA Xilinx Spartan6 LX150. La caméra inclut également divers types de mémoire externe dont deux blocs RAMs de 128 Mo, ainsi qu'une interface Ethernet PHY.

Le FPGA de la caméra contient un système de vision permettant d'intégrer ses propres modules de traitement d'image à l'intérieur.

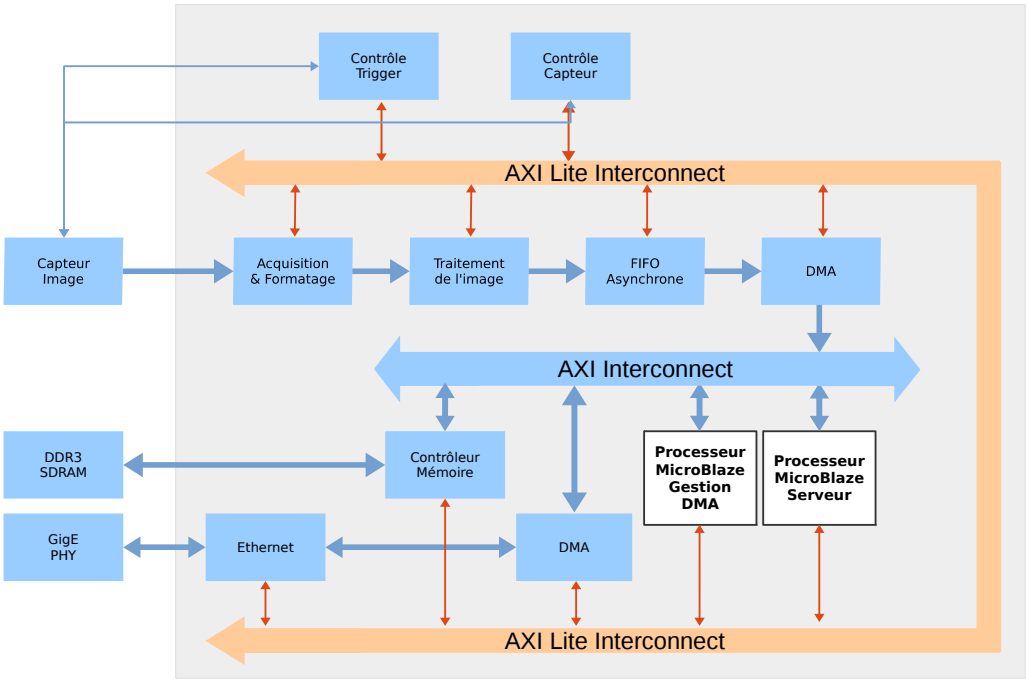


FIGURE 4.2 – Architecture matérielle présente dans le FPGA

4.2.2 Architecture du système de vision

L'architecture matérielle du système de vision présente dans le FPGA de la caméra peut être résumé par la figure 4.2. Cette architecture respecte les principaux points présentés partie 3.4, notamment :

- La capture et la mise en forme du signal transmis par le capteur image.
- Des traitements (facultatifs) bas niveau du signal dans un pipeline.
- La mise en tampon des images traitées dans une RAM externe.
- La transmission des images à un PC client par Ethernet.

Ainsi, cette caméra-FPGA répond à nos besoins.

Le concepteur a fait le choix d'intégrer deux processeurs MicroBlaze dans son système. Le premier processeur permet de faire un contrôle général du système et de gérer la communication Ethernet. Ainsi, le code source de ce CPU contient un serveur permettant de recevoir les commandes du PC client. L'action effectuée par le processeur consiste principalement à lancer ou arrêter l'acquisition, modifier les paramètres des opérateurs de traitement d'image ou du capteur. Le second processeur permet de contrôler les lectures et écritures dans la mémoire externe. Les deux processeurs communiquent via des mailbox.

Les différents composants de ce système de vision sont interconnectés via le protocole AXI4. Les interfaces entre les opérateurs de traitement d'image sont de type Streaming, les interfaces entre les opérateurs et le processeur MicroBlaze

sont de type Lite et les interfaces permettant les lectures et écritures en mémoire externe sont de type Memory Map.

Ces premiers choix impliquent de respecter le format utilisé pour transférer l'image d'une IP à une autre. Il faut donc utiliser le protocole AXI4 pour le transfert des images entre les différentes IPs.

La présence de processeurs MicroBlaze dans le système fait qu'il est possible d'avoir des registres dans les IPs accessibles par le processeur. Ainsi, ces registres peuvent être utilisés pour modifier simplement des paramètres sur les opérateurs développées.

4.2.2.1 Acquisition d'images

L'IP « acquisition et formatage » reçoit les pixels transmis par le capteur via les lignes LVDS et les convertit en un format compatible avec le protocole AXI Streaming. Cette IP s'occupe également du débruitage et de la synchronisation des signaux.

4.2.2.2 Traitements flot de donnée

Cette partie du système est celle où seront insérés les opérateurs de traitement d'image. Ainsi, les traitements se font à la volée, au fur et à mesure du transfert de l'image, ce qui permet des performances temporelles élevées. C'est ici que nous placerons l'opérateur de détection de trait laser.

4.2.2.3 Gestion de la mémoire externe

Après traitement, les résultats sont stockés temporairement en mémoire externe. Étant donné que les fréquences d'horloge sont différentes entre la partie traitement d'image et la partie mise en mémoire, une FIFO asynchrone doit être placée entre ces deux entités.

Afin d'avoir des performances temps réel, l'accès à la mémoire se fait via DMA (direct memory access). Ici, deux entités ont besoin d'accéder à la RAM externe :

- La partie traitement d'image pour l'écriture des résultats en RAM externe.
- La partie communication Ethernet pour lire les données à transmettre vers le PC hôte et stocker temporairement les paquets Ethernet envoyés par le PC hôte.

À cet effet, il faut donc deux DMAs. Ces DMAs sont utilisées en mode « scatter gather » afin de limiter les interventions du processeur. Ce mode de fonctionnement permet d'accéder à différents emplacements de la mémoire externe sans intervention extérieure. Les emplacements mémoire à lire ou écrire sont décrits dans un bloc RAM interne au FPGA. Ceci permet de limiter les interventions du processeur.

L'écriture et la lecture des blocs RAM externes sont réalisées via des contrôleurs mémoire. Le tout est chapeauté par un processeur permettant d'en gérer les lectures et écritures.

4.2.2.4 Communication

Le bloc de communication est constitué d'une IP Ethernet et d'un DMA qui va chercher en mémoire externe les entêtes et les données formant les paquets UDP à transmettre. Dans l'autre sens, le module Ethernet reçoit les paquets transmis par le PC hôte et stocke le contenu des paquet dans la RAM externe du système.

4.3 Contraintes du système

4.3.1 Débit image

Le débit du capteur CMOSIS CMV2000 est disponible dans le tableau 3.2. Ce capteur a un débit maximum de 753 Mpixels/s qu'il est difficile de traiter en temps réel car il faudrait une fréquence d'horloge d'environ 753 MHz pour pouvoir traiter ces images pixel par pixel. Cependant, en mettant en œuvre un parallélisme de données, il est possible de diminuer la fréquence d'horloge sans réduire le débit pixel.

Les fréquences d'horloge minimales nécessaires à ce capteur pour traiter le flot pixel selon le nombre de pixels traités en parallèle sont visibles dans le tableau 3.3. Le nombre de pixels traités en parallèle doit être choisi en fonction de la fréquence maximale que peut atteindre le module le plus lent du système. L'idéal est de choisir un degré de parallélisme permettant une fréquence autour de 100 MHz qui est la fréquence d'horloge de la majorité des systèmes sur FPGA. Dans notre contexte, le parallélisme est de 8 et la fréquence d'horloge est de 95 MHz.

4.3.2 Latence

Il y a environ 100 ms entre le moment où l'objet passe sous le trait laser et le moment où l'actionneur se déclenche. Une latence de 1 à 10 ms est acceptable entre le moment où est acquise l'image et le moment où le résultat est reçu par le PC industriel. Il y a donc peu de contraintes en termes de latence.

4.3.3 Utilisation des signaux Axi Streaming

Le fournisseur de la caméra a décidé de mettre en place son propre protocole pour le transfert de l'image entre les IPs de traitement d'image. Ce protocole est basé sur les signaux du protocole Axi Streaming et ne nécessite pas de signaux supplémentaires pour la synchronisation des lignes et des images. Un exemple de mise en forme des signaux est disponible sur la figure 4.3. Cette exemple est pour une image de taille 4x2 pixels. Le signal *Valid* permet de valider les données et de synchroniser les lignes de l'image. Le signal *Last* permet de synchroniser chaque fin d'image. Le principal inconvénient est qu'une désynchronisation des signaux ou l'absence d'un pixel dans une ligne peut avoir des conséquences importantes sur le traitement effectué.

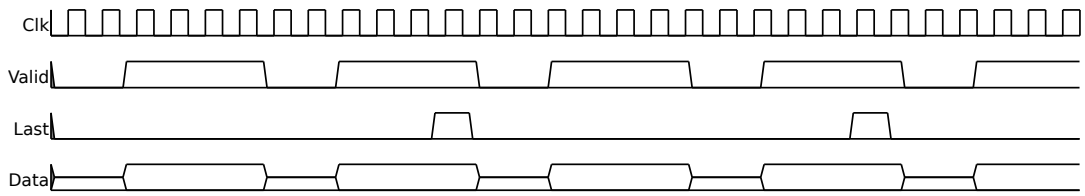


FIGURE 4.3 – Exemple de chronogramme des signaux utilisés pour le transfert d'image entre IPs. Cet exemple est pour une image de résolution 4x2 avec transfert d'1 pixel par cycle d'horloge.

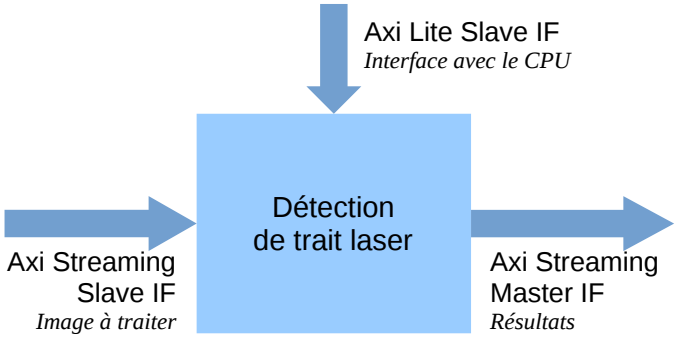


FIGURE 4.4 – Interfaces du module de détection de trait laser

4.4 Architecture matérielle pour la détection de trait laser

Une architecture matérielle a été développée pour la détection de trait laser. Les interfaces de l'IP ainsi que ses paramètres génériques ont été définis. Cette architecture est basée sur l'algorithme « Centre de masse ».

4.4.1 Interfaces de l'IP

L'architecture matérielle développée est composée de trois principales interfaces :

- Une interface Axi Streaming esclave. Il s'agit de l'interface qui recevra les données transmises par le capteur.
- Une interface Axi Streaming maître. Il s'agit de l'interface qui transmettra les résultats de la détection de trait laser.
- Une interface Axi Lite esclave. Cette interface permettra l'interconnexion du module avec le processeur MicroBlaze qui pourra alors modifier la valeur de certains registres dans le module.

Ces interfaces sont représentées sur la figure 4.4.

4.4.2 Interface avec le MicroBlaze

Pour la mise en place de l'interface avec le processeur, nous avons décidé d'utiliser le code source généré par Xilinx lors de la création des IPs. En effet, lors de la création des IPs, il est nécessaire de préciser les interfaces de l'IP. Lorsqu'une interface avec un processeur est mise en place, le code source de l'interface avec le processeur est généré automatiquement.

Nous avons préféré rester sur ce mode de fonctionnement afin d'éviter de développer un code qui serait spécifique à un processeur. Ainsi, si l'on change de processeur, il suffira de générer de nouveau une IP « vide » et de la compléter avec le code source de notre opérateur.

Lors de la création de l'IP, il est nécessaire de préciser le nombre de registres souhaités. Étant donné que nous ne voulions paramétrer que le seuil de détection laser, nous avons choisi d'avoir deux registres 32 bits. Ainsi, si l'on souhaite ajouter un second paramètre (comme le bitshift préalable à la division), il sera possible d'utiliser ce registre supplémentaire pour cela.

4.4.3 Paramètres génériques

Des paramètres génériques ont été sélectionnés pour ce module de détection de trait laser. Les paramètres génériques choisis sont les suivants :

- C_NB_PIXEL : nombre de pixels en parallèle dans le bus de données. Ce paramètre détermine la taille du bus de données avec le paramètre C_PIX_SIZE. Sa valeur est définie selon le débit image et la fréquence d'horloge à respecter. Mieux vaut choisir un nombre qui est multiple de la largeur d'image.
- C_PIX_SIZE : profondeur des pixels ou nombre de bits des pixels. Ce paramètre détermine la taille du bus de données avec le paramètre C_NB_PIXEL. Sa valeur est définie selon le paramétrage du capteur image. Traditionnellement 8, 10, 12 voire 14 bits.
- C_IMG_WIDTH : largeur maximale de l'image. Ce paramètre détermine la longueur des FIFOs avec le paramètre C_NB_PIXEL. Sa valeur est définie par le capteur image.
- C_MAX_IMG_HEIGHT : hauteur maximale de l'image. Ce paramètre permet de déterminer la taille de certains signaux internes. Sa valeur est définie par le capteur image.
- C_LP_POS_SIZE : nombre de bits du résultat de la détection de trait laser. Sa taille est déterminée par le système d'acquisition de données du côté du PC hôte. Traditionnellement 8, 16, 32 voire 64 bits.
- C_THRESHOLD : seuil de détection laser par défaut. Ainsi, la valeur par défaut du seuil peut être définie dès la configuration du système de vision. Sa valeur est déterminée par le paramètre C_PIX_SIZE.

4.4.4 Algorithme

La figure 4.5 représente l'algorithme de détection de trait laser que nous avons décidé de mettre en place. Cet algorithme est constitué de cinq fonctions. Chacune de ces fonctions correspond à une IP qui a été développée puis instanciée dans le module de détection de trait laser.

4.4.4.1 Classification

L'algorithme commence par une classification des pixels où les pixels d'intensité inférieure à un seuil de détection ($C_{THRESHOLD}$) sont mis à zéro. Les résultats sont l'image utile contenant les pixels supérieurs au seuil et un signal indiquant si le pixel correspondant a été seuillé ou non. Pour cette opération, les pixels peuvent être traités indépendamment les uns des autres. Cette fonction est réalisée avec un opérateur de comparaison.

L'entrée de ce module est un bus Axi Stream, dont les données sont les pixels de l'image acquise par le capteur. La taille de ce bus de données est $C_{NB\_PIXEL} \times C_{PIX\_SIZE}$.

La première sortie de ce module est un bus Axi Stream, dont les données sont les pixels de l'image acquise si leur valeur est supérieure au seuil de détection, 0 sinon. La taille de ce bus est $C_{NB\_PIXEL} \times C_{PIX\_SIZE}$. La seconde sortie de ce module est une image binaire. La valeur 1 représente un pixel d'intensité supérieure au seuil de détection. La valeur 0 représente un pixel d'intensité inférieure au seuil de détection. Ce signal est constitué de C_{NB_PIXEL} bits. Un bit pour chaque pixel en parallèle.

La troisième interface de ce module est une interface permettant de définir la valeur du seuil de détection laser. Il est constitué d'un signal correspondant à la nouvelle valeur du seuil et d'un signal indiquant qu'une nouvelle valeur est disponible pour le seuil.

4.4.4.2 Détection nouveau trait laser

La seconde étape de l'algorithme est de détecter le début d'une ligne laser sur l'image. Elle est utile lorsqu'il y a plusieurs traits laser sur une colonne, par exemple, dans un cas de superposition de lignes laser. Cette opération consiste à voir s'il y a un front montant sur la colonne de l'image binaire. Cette opération, facilement réalisable par CPU, est plus complexe à mettre en œuvre lorsque l'on a un flot continu de pixels arrivant par ligne. En effet, lorsque les données arrivent par ligne, il faut mémoriser au moins une ligne de l'image pour effectuer une opération sur une colonne. Cette méthode consistant à mémoriser des lignes d'images est souvent utilisée en traitement d'image sur FPGA, notamment pour des opérations basées sur des noyaux.

Cette mémorisation est effectuée avec un bloc RAM. La profondeur de la RAM est définie par les paramètres C_{NB_PIXEL} et C_{PIX_SIZE}. Sa profondeur est de :

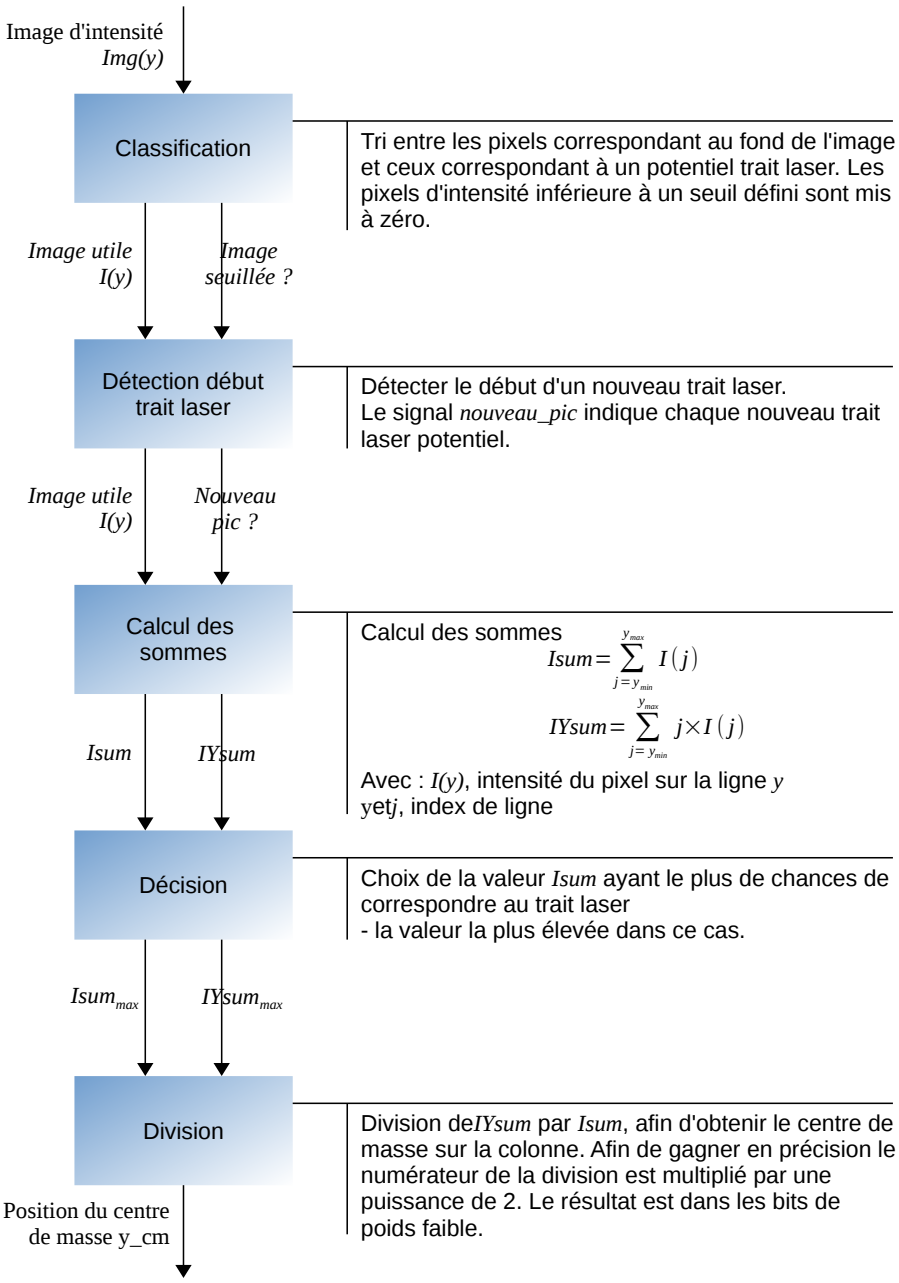


FIGURE 4.5 – Algorithme de détection de trait laser mis en œuvre sur une colonne d'image

$2^{ceil(log_2(C_{IMG\_WIDTH}/C_{NB\_PIXEL}))}$. Étant donné que l'opération est effectuée sur une image binaire, la taille des ports de données du bloc RAM est de C_{NB_PIXEL} bits.

La première entrée de ce module est une interface Axi Stream comprenant l'image utile. La taille du bus de données est de $C_{NB\_PIXEL} \times C_{PIX\_SIZE}$. La seconde entrée de ce module est l'image binaire. La taille de ce signal est de C_{NB_PIXEL} bits.

Ce module possède deux sorties. La première est une interface Axi Stream qui correspond à l'image utile entrant dans le système. La seconde sortie est un signal indiquant le début d'un nouveau trait laser. La taille de ce signal est de C_{NB_PIXEL} bits.

4.4.4.3 Calcul des sommes

Dans cette étape, les sommes S_1 et S_2 de l'algorithme présentées partie 2.3.1 sont calculées. Pour rappel, pour une colonne, ces sommes sont :

$$Isum = \sum_{j=y_{min}}^{y_{max}} I(j) \tag{4.1}$$

$$IYsum = \sum_{j=y_{min}}^{y_{max}} I(j) \times j \tag{4.2}$$

L'architecture choisie pour cette opération consiste à cumuler les intensités des pixels reçus jusqu'à ce que l'on détecte un nouveau pic.

Les résultats intermédiaires sont stockés dans des blocs RAMs en attendant que les pixels de la ligne suivante arrivent. Une machine à états finis a été mise en place pour la synchronisation des lignes durant les étapes de lecture et écriture dans le bloc RAM.

En termes d'interfaces, l'interface d'entrée est constituée d'un bus Axi Stream contenant les pixels utiles. La taille du bus de données est de $C_{NB\_PIXEL} \times C_{PIX\_SIZE}$ bits. L'interface d'entrée contient également un signal indiquant si un nouveau trait laser est détecté. Ce signal est de taille C_{NB_PIXEL} bits, 1 bit par pixel.

La sortie de l'opérateur est constitué des résultats des deux sommes ainsi que des signaux de contrôle, *tvalid* et *tlast* du bus Axi Stream. La taille du signal correspondant à la somme $Isum$ a été calculée en considérant le pire cas possible. Ce cas correspond à une image, de résolution maximale et entièrement blanche. Ainsi, la valeur maximale de $Isum$ est $(2^{C_{PIX\_SIZE}} - 1) \times C_{MAX\_IMG\_HEIGHT}$. Ce résultat tient sur $C_{PIX\_SIZE} + ceil(log_2(C_{MAX\_IMG\_HEIGHT}))$ bits (nombre de bits par pixels + nombre de bits de la hauteur d'image). La valeur maximale de $IYsum$ est :

$$(2^{C_{PIX\_SIZE}} - 1) \times C_{MAX\_IMG\_HEIGHT} \times (1 + C_{MAX\_IMG\_HEIGHT})/2 \tag{4.3}$$

Cette valeur tient sur $C_{PIX\_SIZE} + 2 \times ceil(log_2(C_{MAX\_IMG\_HEIGHT}))$ bits.

4.4.4.4 Décision

Cette étape est importante lorsque des superpositions de traits laser sont présentes. Elle permet de choisir le trait laser correspondant à l'objet. Nous avons décidé que le trait laser dont la somme $Isum$ est la plus importante est l'objet à détecter. Cette étape étant importante pour la précision du système, il pourrait être intéressant à l'avenir d'étudier d'autres critères de choix.

Pour la mise en place de l'opérateur, les mêmes mécanismes que ceux utilisés pour le calcul des sommes et la détection du trait laser ont été utilisés. Ainsi, le maximum de chaque colonne a été obtenu en comparant les données entrantes avec les résultats intermédiaires stockés dans des blocs RAMs.

Les entrées de ce module sont les signaux correspondant aux valeurs de $Isum$ et $IYsum$, ainsi que les signaux de contrôle *tvalid* et *tlast* du bus Axi Stream.

Les sorties de ce module sont les signaux correspondant à la valeur maximale de $Isum$ accompagnée de la valeur $IYsum$ correspondante, ainsi que les signaux de contrôle *tvalid* et *tlast* du bus Axi Stream. La différence avec le module précédent est que les sorties ne sont valides qu'une fois l'image entière traitée. Ainsi, le résultat ne correspond plus qu'à une seule ligne de l'image de départ. Le débit de données en sortie de l'opérateur est plus faible.

4.4.4.5 Division

La dernière étape de l'algorithme consiste à diviser $IYsum$ par $Isum$. Pour cette étape, nous pourrions placer C_{NB_PIXEL} divisions en parallèle et effectuer cette opération au fur et à mesure de la réception des données. Cependant, la division est une opération coûteuse sur FPGA (cf. partie 2.4.1.1). De plus, le débit de données a été diminué lors de l'opération précédente. Il est donc superflu de vouloir gagner du temps sur cette opération. Pour ces deux raisons, nous avons décidé de n'utiliser qu'un seul opérateur de division. Il a donc été nécessaire de sérialiser les données avant l'opération de division et de les dé-sérialiser après.

L'opérateur de division mis en place est une division entière. Avant la division, un décalage à gauche des valeurs du signal $IYsum$ est réalisé afin d'ajouter des bits à ce signal. Ceci permet d'avoir une précision sub-pixel en utilisant la notation en virgule fixe.

La valeur la plus élevée possible pour le résultat de la division est la valeur de $C_{MAX\_IMG\_HEIGHT}$ décalée vers la gauche. Pour une image dont la hauteur maximale est de 1088 et pour laquelle ont été ajouté 8 bits de précision, le résultat le plus élevé possible tient sur 19 bits. Cependant, nous avons décidé que la sortie de la division devait avoir la même taille que le numérateur de la division afin que l'opérateur reste générique. En effet, dans une division entière classique, le résultat le plus élevé apparaît lorsque le dénominateur de la division est égal à 1 et donc, lorsque le résultat est égal au numérateur.

Pour la transmission du résultat au PC hôte, la donnée doit être encapsulée dans une variable dont la taille est dépendante du paramètre $C_{LP\_POS\_SIZE}$. Dans notre contexte, la taille de ce paramètre sera de 16 bits. Le résultat transmis correspondra aux 16 bits de poids faible du résultat de la division.

4.5 Implantation de la détection de trait laser dans le système

L'IP a été placée dans le système de vision avec le logiciel XPS de Xilinx. Cette IP a été placée après l'acquisition de l'image mais avant le transfert de l'image à la mémoire externe. Plus précisément, elle a été placée entre les modules « acquisition et formatage » et « FIFO asynchrone » visibles sur la figure 4.2. L'interface avec les autres IPs de traitement d'image a été fait via le protocole Axi Streaming, et l'interface avec le CPU a été fait avec le protocole Axi Lite. Le bitstream du système a été généré après implantation du module.

Ensuite, l'interfaçage avec le CPU a été réalisé. Les drivers de l'IP développés en C sont ajoutés au projet. Un ou plusieurs registres sont ajoutés auprès du CPU afin d'accéder aux registres du module. Une ou plusieurs commandes sont créées au niveau du serveur afin de permettre la reconfiguration des registres depuis le PC hôte.

4.6 Intégration et validation

La validation de ce système a été effectué étapes par étapes. Dans un premier temps, chaque composant de la détection de trait laser a été simulé. Ensuite, l'IP complète de détection de trait laser a été simulée. Puis l'algorithme a été intégré dans le système de vision, synthétisé, placé et routé et enfin testé sur la caméra-FPGA.

4.6.1 Ressources utilisées

Les ressources utilisées par le système de vision sont obtenues via l'outil XPS de Xilinx. L'utilisation des ressources par l'IP de détection de trait laser ainsi que par ses sous-modules est visible dans le tableau 4.1. L'utilisation des ressources par la caméra-FPGA sans ou avec l'IP de détection de trait laser est visible dans le tableau 4.2.

4.6.2 Essais sur banc de test

Un banc de test a été mis en place pour réaliser un test fonctionnel du système de vision. Ce banc de test représenté sur la figure 4.6 est constitué d'un support laser et d'un support caméra. Un laser ligne rouge a été fixé sur le support, ainsi

TABLE 4.1 – Ressources utilisées par l'IP de détection de trait laser.

Ressource	Slices	Registres	LUTs	MLUTs	BRAMs	DSP48
IP complète	2 284	6 383	5 613	14	44	8
User Logic	2 240	6 324	5 550	14	44	8
Instance LTCM	2 219	6 259	5 548	14	44	8
Classification	31	100	89	0	0	0
Détection début trait	38	189	65	0	1	0
Calcul des sommes	329	1 070	900	0	13	8
Décision	463	1 722	967	0	13	0
Serializer	175	570	238	0	13	0
Division	1 028	2 012	3 228	14	0	0
Deserializer	139	542	59	0	4	0

TABLE 4.2 – Ressources utilisées par le système de vision dans le FPGA.

Ressource	Slices	Registres	LUTs	MLUTs	BRAMs	DSP48
Caméra 16 bits	15 545	39 845	32 113	2 935	144+11	15
%	67 %	21 %	34 %	13 %	53 % + 2 %	8 %
Prototype LT	17 285	45 610	36 760	2 947	144+55	23
%	75 %	24 %	39 %	13 %	53 % + 10 %	12 %

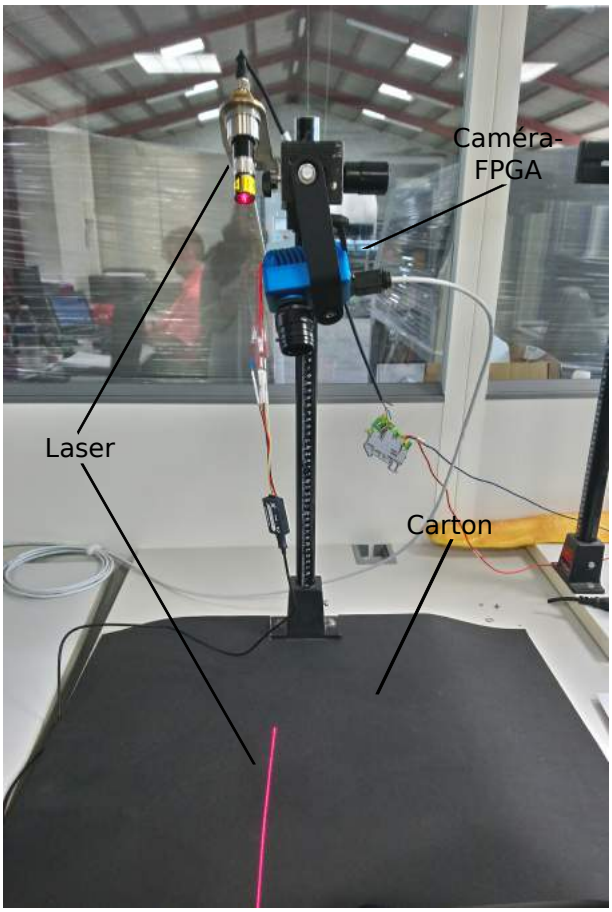


FIGURE 4.6 – Banc de test mis en place

FIGURE 4.7 – Image de trait laser de résolution 2048x128 pixels acquise par le capteur à l'aide du banc de test.

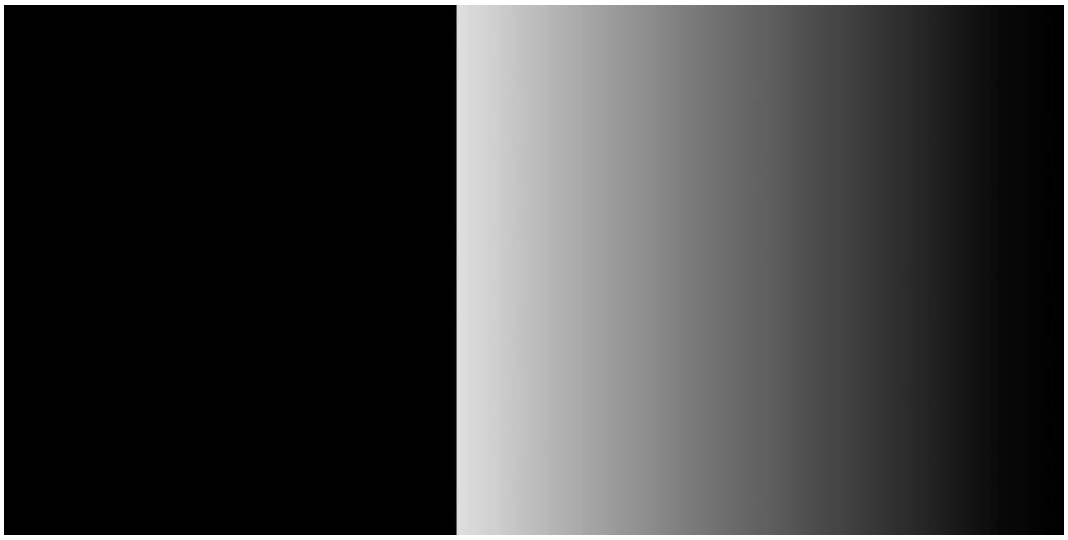


FIGURE 4.8 – Résultat obtenu. Plus l'intensité est élevée, plus le trait laser est bas sur l'image initiale. Les pixels ont été convertis du format 16 bits au format 8 bits afin de pouvoir être visible sur le manuscrit.

que la caméra-FPGA. L'ensemble a été placé sur une table. La ligne laser était projetée à la fois sur la table et sur le sol. La caméra pouvait donc voir le laser sur deux hauteurs très différentes. Comme le support du banc de test était blanc et que la présence d'une lumière ambiante importante pouvaient empêcher la bonne détection du laser, un morceau de carton noir mat a été placé sur la table de façon à ce que la caméra ne voit que le laser.

Le bitstream du système de vision ainsi que les codes sources des CPUs ont été implantés via le programmateur JTAG. Des commandes sont envoyées à la caméra afin d'acquérir des images via un client UDP mis en place lors de la thèse.

La figure 4.7 représente l'image acquise par le capteur à l'aide du banc de test. Cette image de résolution 2048x128 représente la partie du trait laser qui est sur la table. La figure 4.8 représente les images transmises par la caméra après changement de bitstream. On peut voir sur la partie gauche de l'image, en noir, les parties où le trait laser n'a pas été détecté. Sur la partie droite de l'image, on peut voir la position du trait laser sur l'image de départ. Plus l'intensité des pixels est élevée, plus la position du trait laser est basse sur l'image originale.

FIGURE 4.9 – Fausses baguettes.

FIGURE 4.10 – Disques de bois et « bagels ».

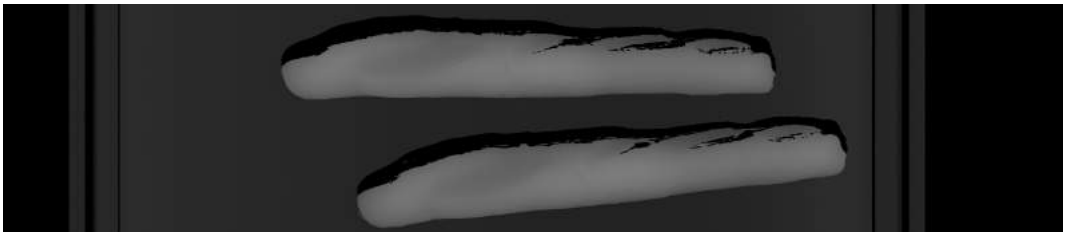


FIGURE 4.11 – Image de fausses baguettes acquise avec le système de vision à une vitesse de 2530 images/s. La résolution initiale est de 2048x5700 pixels. La hauteur a été réduite de 92%

4.6.3 Essais sur une machine

Afin de tester la vitesse d'acquisition maximale du système, ce dernier a été testé sur la boucle test de l'entreprise avec des produits factices. La boucle test est une boucle de convoyeurs avec des machines compteuses ou des machines de contrôle qualité dessus. Cette mise en place permet de tester les systèmes de vision avec des produits avançant à vitesse constante.

La caméra-FPGA a été fixée dans une des machines de test puis programmée via la communication Ethernet. Des images 2,5D de produits ont été acquises à l'aide du client UDP. Différentes valeurs ont été testées pour le framerate. Les produits « mesurés » sont représentés sur les figures 4.9 et 4.10.

Les images reçues correspondent à ce qui était attendu. Des exemples sont disponibles sur les figures 4.11, 4.12 et 4.13. D'après le client UDP, la vitesse d'acquisition maximale atteinte est de 2530 images/s.

4.7 Conclusion

Une caméra-FPGA standard du marché a été prise en main. Cette caméra-FPGA correspond aux critères énoncés dans le chapitre précédent.

Une architecture matérielle générique a été proposée et développée pour la détection de trait laser en respectant les contraintes imposées par la caméra. Ces

FIGURE 4.12 – Disques de bois ⌀120 mm, « bagel », éponge

FIGURE 4.13 – Disques de bois ⌀120 mm, « bagel », fausse baguette

contraintes sont notamment le débit image et la mise en forme de l'image entre les IPs de la caméra-FPGA. Cette architecture matérielle présente des paramètres génériques permettant aux utilisateurs d'adapter cette IP à leur matériel. Ainsi, il est possible d'utiliser un capteur de résolution différente sans modifier l'architecture. Ceci permettra de simplifier le travail des utilisateurs lorsqu'ils concevront de nouveaux systèmes de vision.

Cette architecture a été implantée dans la caméra-FPGA présentée dans ce chapitre. Le système de vision a été vérifié sur banc de test et sa vitesse d'acquisition maximale a été mesurée sur une machine. Ce système a atteint une vitesse de traitement de 2530 images de résolution 2048x128 pixels par seconde. Cette vitesse de traitement est quatre fois supérieure à la vitesse du système de vision présent dans les machines actuelles. Ainsi la principale limitation technique qui était la limitation du débit image due au transfert de l'image entre la caméra et la carte d'acquisition a été levée.

Chapitre 5 : Synthèses et dimensionnements

Sommaire

L'architecture matérielle développée précédemment a été validée et testée pour un modèle de caméra en particulier. L'objectif de cette partie est de prédire l'utilisation des ressources de cette architecture matérielle suite à un changement de matériel, notamment à un changement de capteur ou de FPGA.

Pour cela, nous allons estimer les ressources nécessaires à l'opérateur de détection de trait laser selon les valeurs des paramètres génériques. En effet, un changement de capteur influence les valeurs de paramètres de largeur et de hauteur d'image. Il est donc important d'estimer l'influence de la modification de ces paramètres sur les ressources utilisées. De plus, une augmentation du débit de données peut amener à augmenter le nombre de pixels traités en parallèle. Il est important de savoir si le FPGA utilisé a la capacité de contenir l'IP de détection de

trait laser malgré une augmentation du nombre de pixels en parallèle. Cela permet d'estimer quel FPGA devrait être utilisé.

5.1 Principe

Le dimensionnement des ressources consiste à prédire l'utilisation des ressources selon le paramètre choisi en définissant un modèle mathématique de l'évolution des ressources utilisées selon ce paramètre. Cela permet d'anticiper les besoins en termes de matériel selon la résolution du capteur ou le débit pixel à traiter. Ainsi, à partir de ces modèles mathématiques, l'entreprise peut évaluer approximativement les besoins en termes de ressources du FPGA et être guidée dans le choix d'un autre FPGA.

Ces explorations de l'espace des paramètres permettent également d'étudier le comportement de l'outil lorsque peu de ressources sont disponibles dans le FPGA. Les différents comportements observés selon le pourcentage d'utilisation des ressources permettent de définir des classes de modèles. La première classe est celle correspondant au cas où suffisamment de ressources sont disponibles dans le FPGA. C'est le cas le plus courant. Dès qu'un changement de comportement important est visible, une nouvelle classe de modèle peut être définie. Ces changements de comportement apparaissent souvent lorsque peu de ressources sont disponibles. Cela permet de tester les limites de la technologie FPGA. En raison de cette étude, les synthèses ont parfois été réalisées pour des valeurs élevées de paramètres bien que ces valeurs ne sont pas actuellement utilisées en pratique.

5.2 Méthodologie

Nous avons dimensionné les ressources utilisées par l'IP en se basant sur le travail effectué les auteurs de [FCPR15]. Ces travaux présentent une méthodologie pour la mise en place de modèles mathématiques pour le dimensionnement pour les NoC. Nous avons décidé d'utiliser la même méthodologie pour le dimensionnement de notre IP. Le dimensionnement se fait en quatre étapes :

- La collection de données.
- L'analyse de données.
- La modélisation des données.
- La validation des modèles.

L'étape de collection de données consiste à effectuer des synthèses en faisant varier certains paramètres et en laissant d'autres paramètres fixes. Les résultats d'utilisation des ressources contenues dans le rapport de synthèse sont extraites. Les ressources observées sont les registres, les LUTs, les blocs DSP et les blocs RAM. Les LUTs peuvent être divisées en deux groupes : les LUTs logiques que nous appelleront LLUTs, et les LUTs mémoire que nous appelleront MLUTs. Ici, les synthèses ont été effectuées avec l'outils XST 13.3 pour les FPGAs Spartan6

TABLE 5.1 – Valeurs par défaut des paramètres de l'IP pour le dimensionnement de l'IP

Paramètre	Valeur par défaut	Plage de variation
C_{NB_PIXEL}	8	$n \geq 1$, multiple de C_{IMG_WIDTH}
C_{PIX_SIZE}	10	$32 \geq n \geq 1$
C_{IMG_WIDTH}	2048	$n \geq 1$, multiple de 8
$C_{MAX\_IMG\_HEIGHT}$	1088	$n \geq 1$
$C_{LP\_POS\_SIZE}$	16	8, 16 ou 32
$C_{THRESHOLD}$	128	$n \leq 2^{C_{PIX\_SIZE}} - 1$

LX150, Virtex6 LX195T, et Virtex7 X330T. Les valeurs par défaut des paramètres génériques lors des synthèses est disponible dans le tableau 5.1.

L'étape d'analyse de données consiste à effectuer des corrélations entre l'utilisation des ressources et les valeurs des paramètres. À partir de ces corrélations, il est possible de voir comment ces données sont liées entre elles. Cela permet de savoir s'il est possible d'extraire un ou plusieurs modèles dans le comportement des données.

L'étape de modélisation consiste à extraire un ou plusieurs modèles mathématique pour l'utilisation des ressources selon le paramètre lorsque cela est possible. Cela s'effectue notamment via des interpolations ou approximations.

La validation des modèles consiste à vérifier la pertinence du ou des modèles. Cette étape permet également de mettre en avant les limites de la modélisation. Les modèles sont validés via le calcul de l'erreur relative entre les ressources utilisées après synthèses et la valeur estimée via le modèle mathématique.

5.3 Nombre de pixels en parallèle

Le premier paramètre étudié est le nombre de pixels en parallèle.

5.3.1 Collection des données

L'IP a été synthétisée pour 3 familles de FPGA différentes. Pour ces synthèses, nous avons fait varier le nombre de pixels en parallèle en gardant les autres paramètres fixes. Les valeurs des autres paramètres génériques sont les valeurs par défaut représentées dans le tableau 5.1. Des synthèses ont été effectuées pour 50 valeurs du nombre de pixels en parallèle comprises entre 1 et 400. En pratique, nous n'irons jamais jusqu'à des valeurs aussi élevées. L'idée est de voir le comportement de l'outil de synthèse lorsqu'il y a peu de ressources dans le FPGA. L'utilisation des registres, LUTs (LLUTs et MLUTs), blocs DSP, et blocs RAM est représentée sur les figures 5.1 et 5.2. Les modèles de dimensionnement proposés partie 5.3.3 sont également représentés sur ces figures 5.1 et 5.2.

TABLE 5.2 – Ressources utilisées et fréquence maximale pour cinq valeurs du paramètre « nombre de pixels en parallèle »

Val. Paramètre	8	16	32	48	64
Registres	6259	10198	18039	25890	37795
LUTs (total)	5910	7913	11936	15993	22529
LLUTs	5896	7899	11922	15979	19827
MLUTs	14	14	14	14	2702
BRAMs	22	42	80	119	101
DSP48	8	16	32	48	64
Période min. (ns)	7,469	7,513	7,513	7,513	7,513
Fréq. max. (MHz)	133,888	133,101	133,101	133,101	133,101

TABLE 5.3 – Paramètres associés aux données V1 à V7

V1	V2	V3	V4	V5	V6	V7
C_{NB_PIXEL}	Registres	LUTs (total)	LLUTs	MLUTs	BRAMs	Blocs DSP

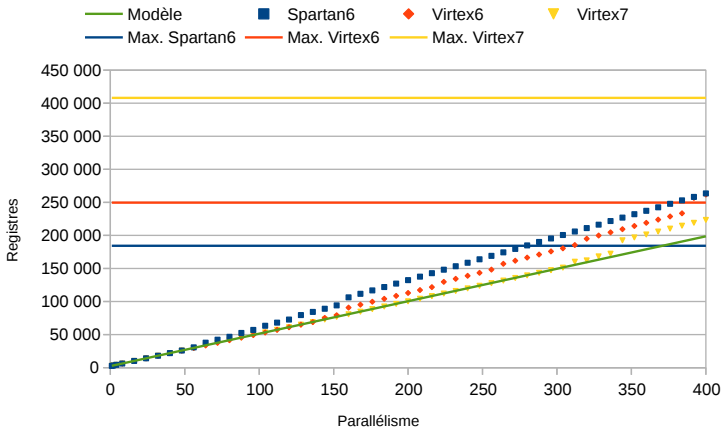
Les ressources utilisées par le module de détection de trait laser pour cinq de ces valeurs pour le FPGA Spartan6 sont visibles dans le tableau 5.2.

5.3.2 Analyse des données

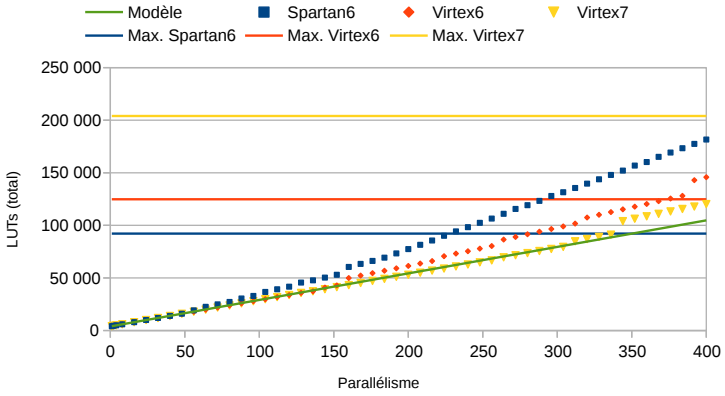
Les données collectées précédemment sont analysées via le logiciel R. Ce logiciel calcule les corrélations entre les données. Ici les données étudiées sont les ressources consommées par l'IP. Les paramètres correspondants aux données étudiées sont représentés dans le tableau 5.3. La figure 5.3 a été obtenue via ce logiciel. Elle représente les corrélations entre les 7 données disponibles. Les tableaux 5.4 et 5.5 représentent les coefficients de corrélation entre les données V1 à V7 pour les FGPAs Spartan6 et Virtex7. Les coefficients de corrélation obtenus pour ces deux FPGAs sont similaires. Cela signifie qu'il est possible de proposer des modèles en se basant sur les données d'un seul FPGA.

Nous pouvons remarquer sur la figure 5.3 qu'il y a une forte corrélation entre les ressources, en dehors des blocs RAM. En effet, les coefficients de corrélation calculés par R sont pour les colonnes V1 à V5 supérieurs à 0,99. Cela signifie qu'il y a une interdépendance entre ces paramètres et qu'il est possible d'en extraire des modèles mathématiques. Sur les figures 5.1 et 5.2, nous pouvons également observer que certaines droites sont « brisées » ce qui signifie que plusieurs modèles linéaires peuvent être définis.

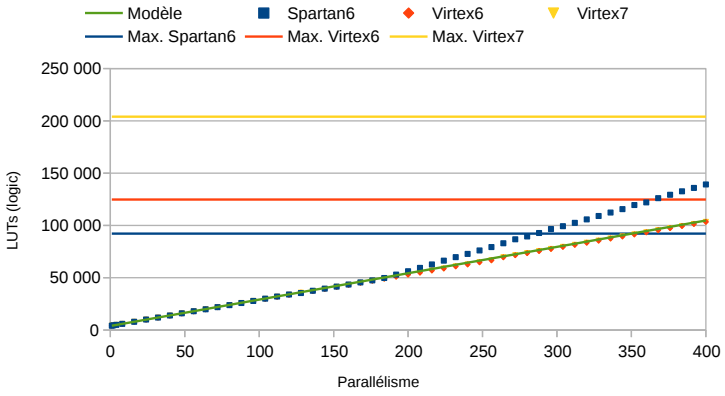
Sur la colonne V7 qui correspond au nombre de blocs DSP utilisés, nous pouvons voir que le coefficient de corrélation est plus faible : entre 0,81 et 0,88. Cependant, lorsque l'on observe les figures, nous observons que la consommation de



(a) Registres

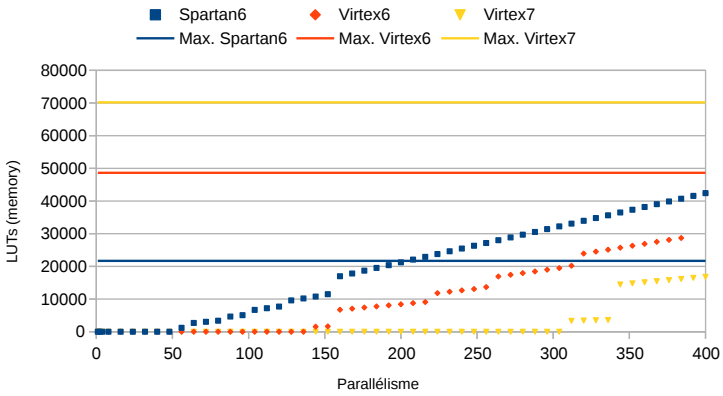


(b) LUTs

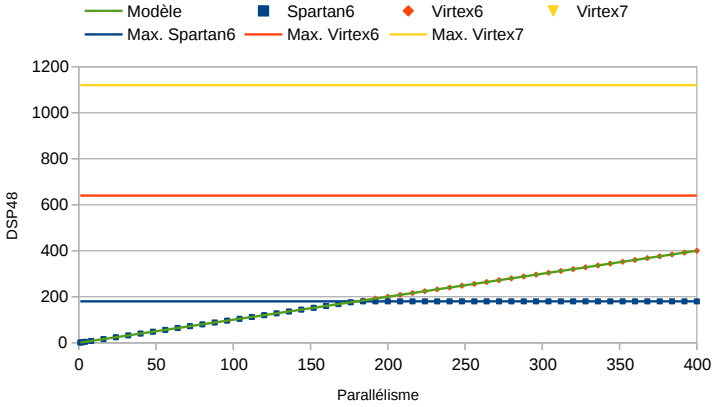


(c) LLUTs

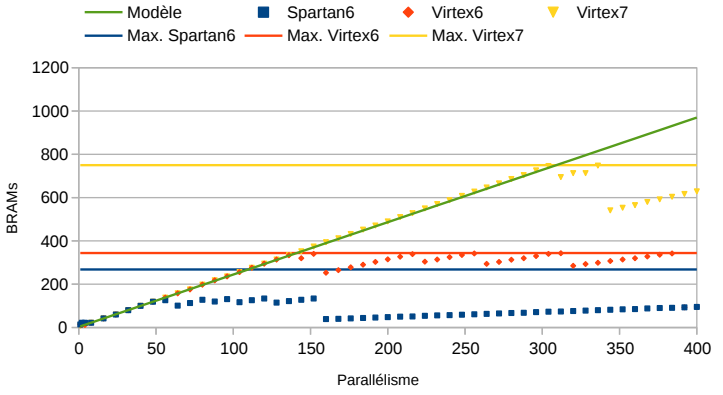
FIGURE 5.1 – Utilisation des ressources selon le nombre de pixels en parallèle pour trois familles de FPGA et modèles associés obtenus partie 5.3.3 avec les données du Spartan6.



(a) MLUTs



(b) Blocs DSP



(c) Blocs RAM

FIGURE 5.2 – Utilisation des ressources selon le nombre de pixels en parallèle pour trois familles de FPGA et modèles associés obtenus partie 5.3.3 avec les données du Spartan6 (suite).

TABLE 5.4 – Corrélations entre les données V1 à V7 pour le FGPA Spartan6 pour le paramètre « nombre de pixels en parallèle ».

	V1	V2	V3	V4	V5	V6	V7
V1	1.000000000	0.99954126	0.99542470	0.992561262	0.9935057	0.003792754	0.877776766
V2	0.999541256	1.00000000	0.99608096	0.992422373	0.9964827	-0.026253575	0.875815528
V3	0.995424701	0.99608096	1.00000000	0.999099478	0.9924295	-0.022989783	0.829904300
V4	0.992561262	0.99242237	0.99909948	1.000000000	0.9863304	0.006730315	0.812990605
V5	0.993505661	0.99648268	0.99242946	0.986330400	1.0000000	-0.109593351	0.870035460
V6	0.003792754	-0.02625357	-0.02298978	0.006730315	-0.1095934	1.000000000	0.002237405
V7	0.877776766	0.87581553	0.82990430	0.812990605	0.8700355	0.002237405	1.000000000

TABLE 5.5 – Corrélations entre les données V1 à V7 pour le FGPA Virtex7 pour le paramètre « nombre de pixels en parallèle ».

	V1	V2	V3	V4	V5	V6	V7
V1	1.0000000	0.9132646	0.9168086	0.9175168	0.9126657	0.1521419	0.8646508
V2	0.9132646	1.0000000	0.9986591	0.9974686	0.9999825	-0.1623471	0.9703087
V3	0.9168086	0.9986591	1.0000000	0.9998123	0.9983374	-0.1552886	0.9565047
V4	0.9175168	0.9974686	0.9998123	1.0000000	0.9970331	-0.1525199	0.9506769
V5	0.9126657	0.9999825	0.9983374	0.9970331	1.0000000	-0.1631804	0.9717172
V6	0.1521419	-0.1623471	-0.1552886	-0.1525199	-0.1631804	1.0000000	-0.1871687
V7	0.8646508	0.9703087	0.9565047	0.9506769	0.9717172	-0.1871687	1.0000000

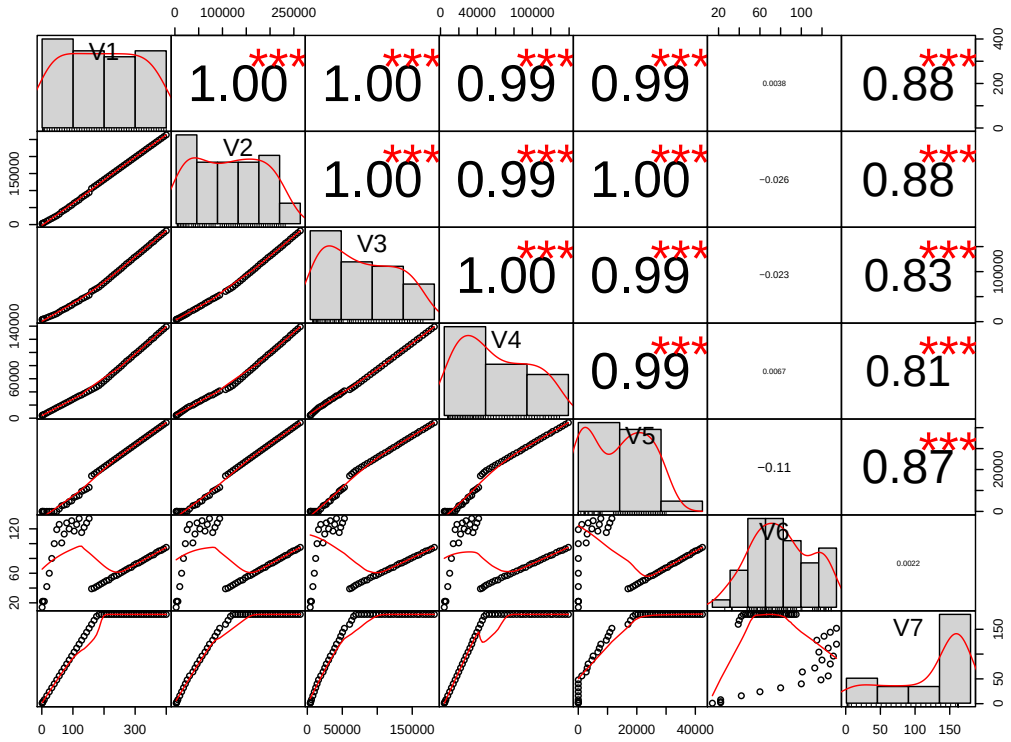


FIGURE 5.3 – Corrélations entre les valeurs du paramètre et les ressources utilisées pour les données du FPGA Spartan6. Figure obtenue avec le logiciel R pour le paramètre « nombre de pixels en parallèle ».

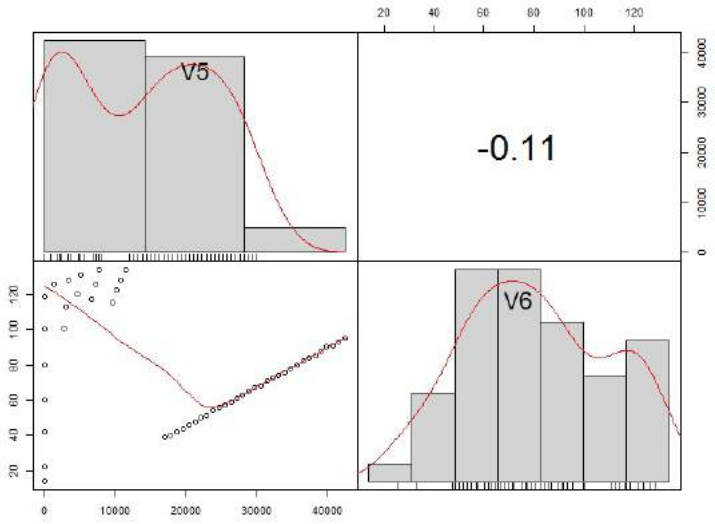


FIGURE 5.4 – Corrélations entre l'utilisation des MLUTs (V5) et des BRAMs (V6) pour les données du FPGA Spartan6. Figure obtenue avec le logiciel R pour le paramètre « nombre de pixels en parallèle ».

blocs DSP augmente de façon linéaire jusqu'au point où celle-ci sature. Il y a donc présence de deux zones de linéarité pour cette ressource. Cela signifie que deux modèles mathématiques différents peuvent être extraits pour cette ressource : un pour la zone où la consommation de blocs DSP augmente de façon linéaire, un pour la zone où la consommation de blocs DSP est constante (saturation du niveau d'utilisation de la ressource).

La ressource qui a le comportement le moins corrélé avec les autres ressources est la BRAM. Cependant, en observant les figures 5.3 et 5.4, nous pouvons observer trois zones présentant une linéarité dans l'utilisation des BRAMs. Cela indique la présence de corrélations entre l'utilisation des BRAMs et des MLUTs et entre l'utilisation des BRAMs et le niveau de parallélisme malgré la faible valeur du coefficient de corrélation. Ces trois zones de linéarité sont également visibles sur la figure 5.2 pour les familles Spartan6 et Virtex6. Il est donc possible d'extraire un modèle d'utilisation des BRAMs dans chacune de ces zones.

Pour conclure, le paramètre C_{NB_PIXEL} présente plusieurs plages de parallélisme dans lesquelles un modèle mathématique peut être défini pour l'utilisation des ressources. Ces différentes plages sont utilisées pour définir des classes de modèles. Ainsi pour le FPGA Spartan6, la première classe de modèles est définie lorsque le paramètre a une valeur comprise entre 1 et 60. La première valeur (1) correspond à la valeur minimale du paramètre. La seconde valeur correspond à la valeur à partir de laquelle, 50 % des BRAMs sont utilisés. La seconde classe de modèle du Spartan6 peut être définie lorsque les valeurs du parallélisme sont comprises entre 60 et 154. La valeur 154 correspond à la valeur à partir de laquelle l'outil de synthèse préfère utiliser des MLUTs et des registres plutôt que des BRAMs. La

troisième classe de modèle peut définie pour des valeurs de parallélisme comprises entre 154 et 180. La valeur 180 correspond à la valeur du paramètre à partir de laquelle 100 % des blocs DSPs sont utilisés. Enfin, une quatrième classe peut être définie lorsque les valeurs du paramètre sont supérieures à 180.

Pour les FPGAs Virtex6 et Virtex7, la première classe se termine pour la valeur du paramètre à partir duquel 100 % des BRAMs sont utilisés, respectivement 140 et 308. Enfin, pour le FPGA Virtex6, la seconde classe se termine pour la valeur 388. L'évaluation du Virtex7 ne présente pas cette limite car il faudrait effectuer plusieurs dizaines de synthèses en plus pour en estimer la valeur. Une estimation approximative de cette valeur limite peut être obtenue via le calcul suivant : $(N_{BRAM\_total} \times T_{BRAM\_kb})/32$ avec N_{BRAM_total}, le nombre total de BRAMs dans le FPGA et T_{BRAM_kb} la taille d'un bloc RAM en kb. Avec cette formule, on peut estimer que la limite du Virtex7 sera atteinte pour un parallélisme autour de 844.

5.3.3 Modélisation

Dans la première zone de linéarité, une régression linéaire a été effectuée pour extraire les modèles d'utilisation des ressources. Cette régression linéaire a été effectuée avec les données du Spartan6 pour cinq valeurs de parallélisme comprises entre 8 et 48. Les modèles de classe 1 pour chacune des ressources sont :

- Registres : $f(x) = 490,6483x + 2339,6441$
- LUTs (total) : $f(x) = 252,0678x + 3884,2373$
- LLUTs : $f(x) = 252,0678x + 3870,2373$
- MLUTs : $f(x) = 14$
- BRAMs : $f(x) = 2,4174x + 2,8983$
- DSPs : $f(x) = x$

Ces modèles de classe 1 sont représentés avec l'évolution des ressources dans les figures 5.1 et 5.2. Nous pouvons observer que les modèles linéaires sont en adéquation avec les données obtenues dans la première classe. De plus, ces modèles sont définis pour les trois familles de FPGA dans la classe 1. Dans les plages de parallélisme correspondant aux autres classes, une différence apparaît entre les modèles et les données associées.

À partir du modèle proposé pour l'évolution des blocs DSPs, nous pouvons déduire la valeur délimitant les classes 3 et 4 pour les FPGAs Virtex6 et 7. Les valeurs délimitant ces classes sont respectivement 640 pour le Virtex6 et 1120 pour le Virtex7. Le tableau 5.6 est un résumé des valeurs de parallélisme définissant les classes de modèle. Les valeurs de ce tableau ne sont valables que pour la synthèse de l'IP seule. Si l'on souhaite insérer l'IP dans un système existant, il faut prendre en compte le nombre de ressources restantes après implantation du système existant.

Comme décrit dans la section précédente, l'utilisation des ressources augmente linéairement jusqu'à atteindre une première limite. Cette première limite de linéarité correspond au moment où l'outil de synthèse décide d'optimiser l'utilisation des blocs RAM. Ainsi, les blocs RAMs sont remplacés par des MLUTs. Une seconde limite de linéarité apparaît lorsqu'il n'y a plus de blocs DSP disponibles. Les

TABLE 5.6 – Définition des classes de modèles suivant la famille de FPGA

	Classe 1	Classe 2	Classe 3	Classe 4
Spartan6	[1 - 60]	[60 - 154]	[154 - 180]	[180+]
Virtex6	[1 - 140]	[140 - 388]	[388 - 640]	[640+]
Virtex7	[1 - 308]	[308 - 844 (est.)]	[844 (est.) - 1120]	[1120+]

blocs DSPs sont donc remplacés par des registres et des LUTs, ce qui en augmente la consommation.

5.3.4 Validation du modèle

L'erreur relative peut être calculée entre l'approximation linéaire et les ressources réellement utilisées. Cette erreur relative a été calculée pour les registres et les LUTs. Elle est représentée sur les figures 5.5 et 5.6.

Les limites de linéarité et donc les différentes classes définies sont clairement visibles sur ces figures 5.5 et 5.6. Nous pouvons observer l'augmentation de l'utilisation des registres et MLUTs à partir de 60 (Spartan6),140 (Virtex6) et 308 (Vitex7) pixels en parallèle. Chaque palier horizontaux sur les figures représente une baisse de l'utilisation des blocs RAM et une hausse de l'utilisation des registres et des MLUTs. Pour le FPGA Spartan6, nous pouvons voir sur la figure 5.5c la limite de linéarité pour l'utilisation des blocs DSP (180) et donc l'augmentation de l'utilisation des LLUTs. Dès que le nombre maximum de blocs DSP est utilisé, l'outil de synthèse utilise les LUTs du FPGA pour réaliser les fonctions DSP. Ainsi, est visible sur la figure l'augmentation régulière du nombre de LUTs utilisées après l'abscisse 180.

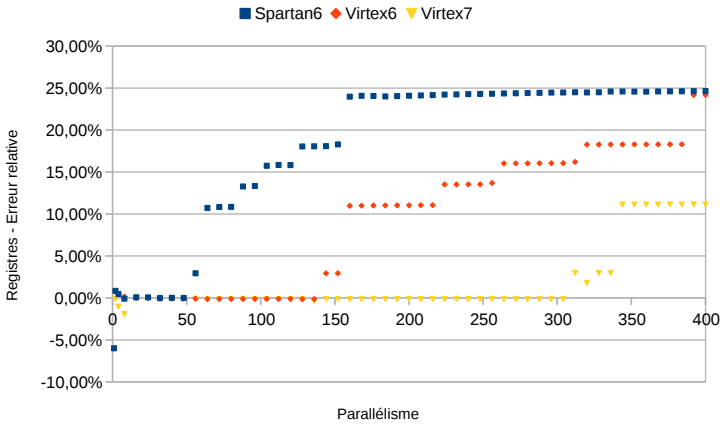
Dans la première zone de linéarité, nous pouvons observer des erreurs relatives inférieures à 5 %, excepté pour la première synthèse effectuée avec la famille Spartan6.

Nous pouvons également remarquer que le modèle défini pour le FPGA Spartan6 est valable pour les FPGAs Virtex 6 et 7 dans la première zone de linéarité.

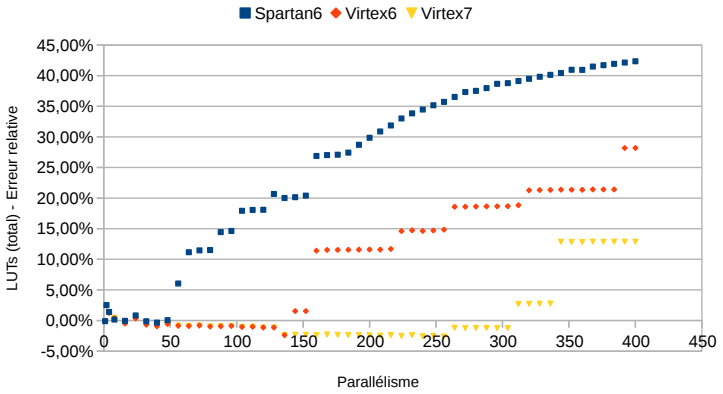
5.3.5 Conclusion

Des modèles linéaires ont pu être extraits pour chacune des ressources selon le nombre de pixels en parallèle. Ces modèles sont valides tant qu'aucune autre ressource du FPGA ne manque. Nous les définissons donc comme étant des modèles de classe 1. Ainsi dès qu'une ou plusieurs ressources viennent à manquer dans le FPGA, une nouvelle classe de modèles peut être définie. Ces modèles de classe 2, 3 ou plus doivent cependant être redéfini pour chaque famille de FPGA contrairement au modèle de classe 1.

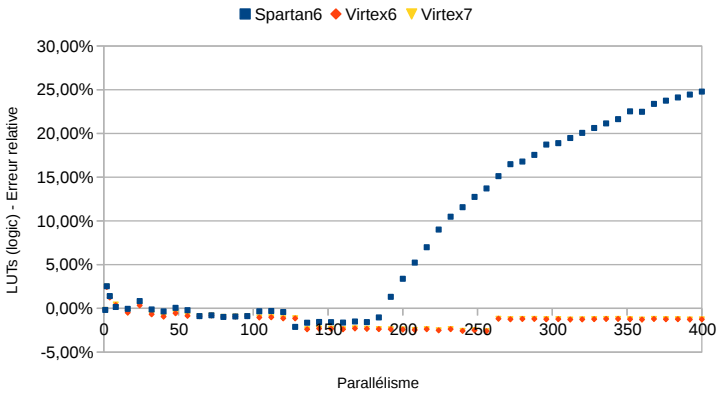
Les ressources limitantes pour cette IP sont dans un premier temps les blocs RAM, puis les blocs DSP. Lorsqu'il n'y a plus de blocs RAM disponible, l'outil de



(a) Registres

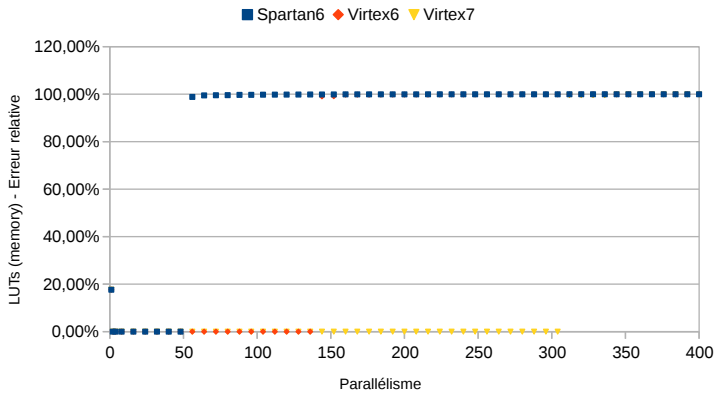


(b) LUTs

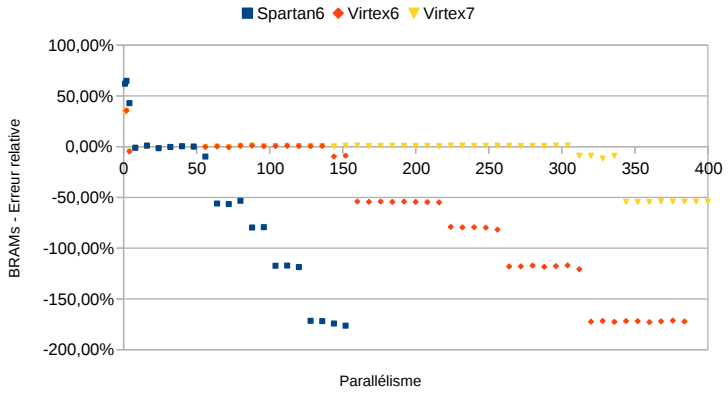


(c) LLUTs

FIGURE 5.5 – Erreur relative entre le modèle d'utilisation des ressources selon le nombre de pixels en parallèle et les valeurs réelles pour trois familles de FPGA.



(a) MLUTs



(b) BRAMs

FIGURE 5.6 – Erreur relative entre le modèle d'utilisation des ressources selon le nombre de pixels en parallèle et les valeurs réelles pour trois familles de FPGA (suite).

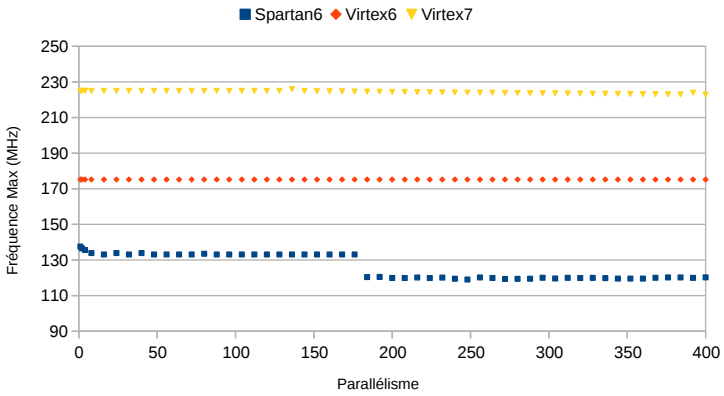


FIGURE 5.7 – Fréquence maximale après synthèse selon le nombre de pixels en parallèle pour trois familles de FPGA.

synthèse les remplace par des MLUTs. Cependant les MLUTs sont également en nombre limité. Pour le Spartan6, l'utilisation des blocs RAM est limitée à 50 % des ressources tandis que pour les Virtex 6 et 7, cette limitation est à 90 %. L'utilisation des blocs RAM est donc différente malgré l'utilisation du même outil de synthèse. Lorsque les blocs DSP manquent, ils sont implantés sur des LLUTs. Plus globalement, nous pouvons remarquer que les ressources mémoire, qu'il s'agisse des blocs RAM ou des MLUTs sont critiques sur FPGA.

Aujourd'hui, le parallélisme de l'IP est de 8, cependant, si l'on souhaite utiliser des capteurs image plus performants, il faudrait envisager de pouvoir monter à un niveau de parallélisme de 32 voire 64 pixels (voir tableau 3.3). Un parallélisme de 64 tomberait dans la deuxième classe de modèle du Spartan6. L'étude du comportement de l'utilisation des ressources dans cette classe permettrait de prévoir, s'il reste envisageable d'implanter cette IP dans un système de vision pour ce FPGA en particulier ou s'il faudrait envisager de passer à une autre famille de FPGA.

En termes de fréquences d'horloge, la fréquence maximale de l'IP varie faiblement. La principale différence se situe entre les différentes familles de FPGA. Pour le Spartan6, il y a un décrochement qui correspond au moment où tous les blocs DSP du FPGA sont utilisés.

5.4 Largeur d'image

Le second paramètre étudié est la largeur d'image. Il influence principalement la profondeur des BRAMs/FIFOs utilisées dans l'IP.

5.4.1 Collection des données

L'IP de détection de trait laser a été synthétisée pour environ 30 valeurs de largeur d'image comprises entre 1016 et 65536. Nous avons choisi ces valeurs afin

TABLE 5.7 – Ressources utilisées et fréquence maximale pour cinq valeurs du paramètre largeur d'image

Val. Paramètre	2048	3072	4096	4104	8192
Registres	6275	6287	6287	6299	6299
LUTs (total)	5939	5963	5953	5976	5965
LLUTs	5925	5949	5939	5962	5951
MLUTs	14	14	14	14	14
BRAMs	22	42	42	80	80
DSP48	8	8	8	8	8
Période min. (ns)	7,469	7,469	7,469	7,469	7,469
Fréq. max. (MHz)	133,888	133,888	133,888	133,888	133,888

TABLE 5.8 – Paramètres associés aux données V1 à V6 pour l'étude du paramètre « largeur d'image ».

V1	V2	V3	V4	V5	V6
C_{IMG_WIDTH}	Registres	LUTs (total)	LLUTs	MLUTs	BRAMs

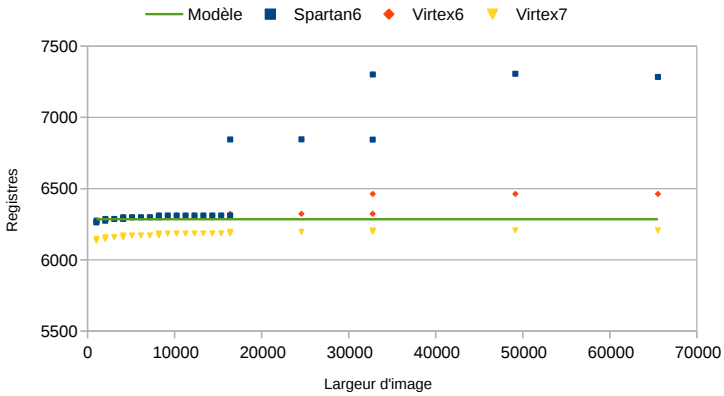
d'avoir, d'une part, un nombre élevé de paliers permettant de vérifier le modèle. D'autre part, cela permettra d'observer le comportement de l'outil de synthèse lorsque peu de ressources sont disponibles. Ces synthèses ont été effectuées pour trois familles de FPGA. Les valeurs des autres paramètres génériques sont représentées dans le tableau 5.1.

Les ressources utilisées par l'IP de détection de trait laser ainsi que sa fréquence maximale pour cinq largeurs d'image sont visibles dans le tableau 5.7. Les résultats de ces synthèses pour les registres, LUTs (LLUTs et MLUTs) et blocs RAM sont disponibles sur les figures 5.8 et 5.9. L'utilisation des blocs DSP est constante. De même, la fréquence maximale est constante quelque soit la largeur de l'image.

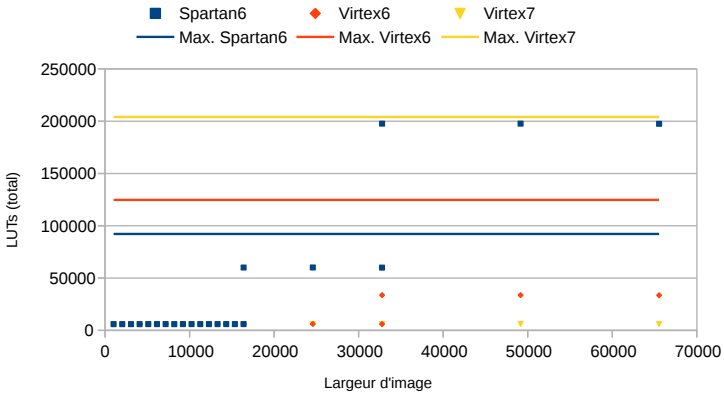
5.4.2 Analyse des données

Les tableaux contenant les ressources utilisées ont été analysées avec le logiciel R. L'utilisation des blocs DSP étant constante, cette variable n'a pas été intégrée dans l'analyse. Les paramètres correspondants aux données étudiées sont représentés dans le tableau 5.8. La figure 5.10 a été obtenue via ce logiciel et représente les corrélations entre les 6 données observées. Le tableau 5.9 représente les coefficients de corrélation entre les données V1 à V6 pour le FPGA Spartan6.

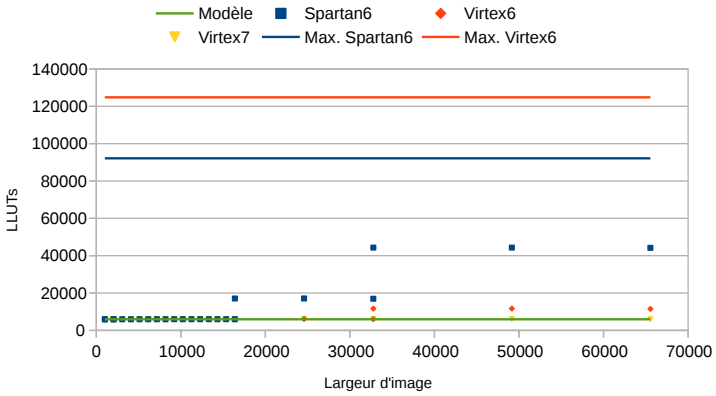
Les variables ne sont pas assez homogènes dans la plage de valeur pour être analysées correctement. Cependant, cela donne un premier aperçu. Les coefficients de corrélation ne sont pas aussi élevés que pour le nombre de pixels en parallèle. Néanmoins,nous pouvons observer des corrélations entre l'utilisation des registres



(a) Registres

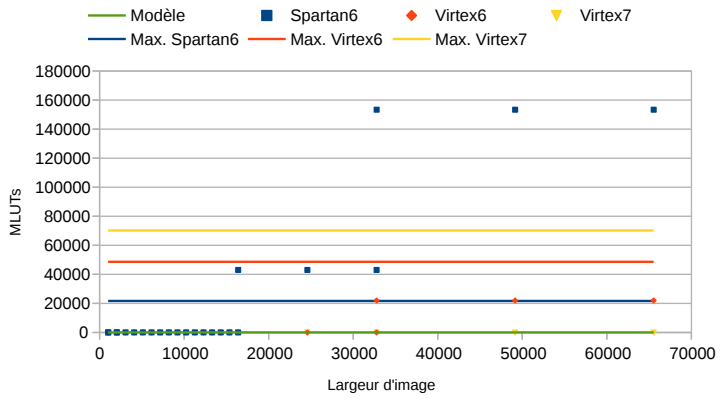


(b) LUTs

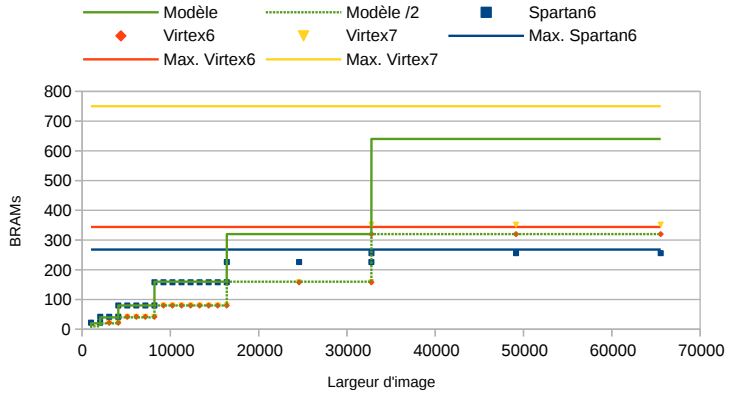


(c) LLUTs

FIGURE 5.8 – Utilisation des ressources selon la largeur d'image pour trois familles de FPGA et modèles associés obtenus partie 5.4.3 avec les données du Spartan6.



(a) MLUTs



(b) Blocs RAM

FIGURE 5.9 – Utilisation des ressources selon la largeur d'image pour trois familles de FPGA et modèles associés obtenus partie 5.4.3 avec les données du Spartan6 (suite).

TABLE 5.9 – Corrélations entre les données V1 à V6 pour le FGPA Spartan6 pour le paramètre « largeur d'image ».

	V1	V2	V3	V4	V5	V6
V1	1.0000000	0.9005635	0.8916260	0.8920864	0.8914990	0.8279172
V2	0.9005635	1.0000000	0.9690727	0.9707434	0.9686450	0.7778919
V3	0.8916260	0.9690727	1.0000000	0.9999744	0.9999984	0.6927150
V4	0.8920864	0.9707434	0.9999744	1.0000000	0.9999600	0.6955776
V5	0.8914990	0.9686450	0.9999984	0.9999600	1.0000000	0.6919826
V6	0.8279172	0.7778919	0.6927150	0.6955776	0.6919826	1.0000000

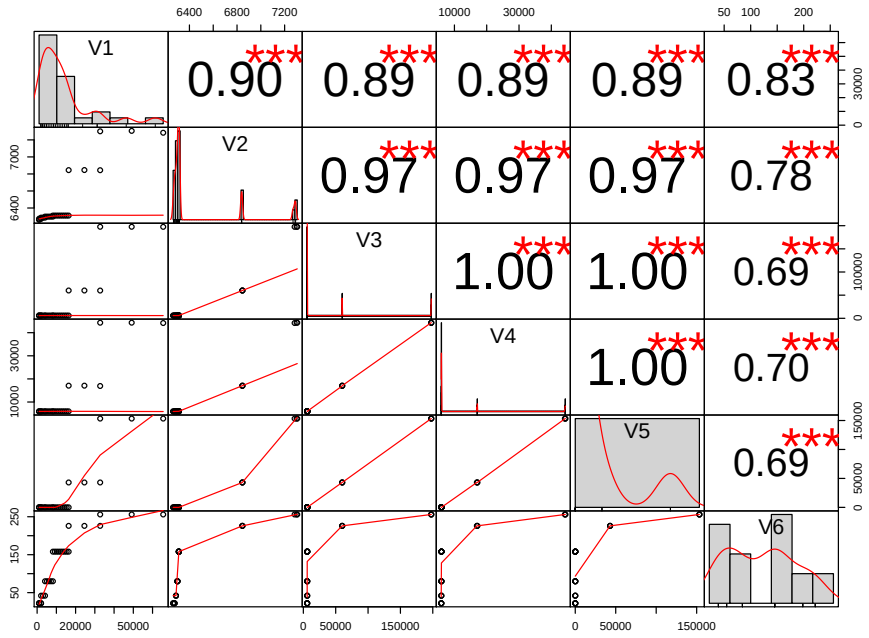


FIGURE 5.10 – Corrélations entre les valeurs du paramètre et les ressources utilisées pour les données du FPGA Spartan6. Figure obtenue avec le logiciel R pour le paramètre « largeur d'image ».

et des LUTs. Nous pouvons également relever sur la figure les liens entre l'utilisation des BRAMs et des MLUTs. Pour les ressources V2 à V5, nous pouvons observer une première zone avec une très faible utilisation des ressources, puis une zone dans laquelle l'utilisation des ressources augmente de façon importante. Cette augmentation de la consommation des ressources est linéaire entre les registres et les LUTs. En parallèle, nous pouvons noter pour la ressource V6 (BRAMs) une hausse importante de l'utilisation de cette ressource puis un plafonnement de son utilisation.

La présence de corrélations nous permet d'affirmer qu'il est possible d'extraire des modèles. Cependant, le modèle d'évolution des données n'étant pas linéaire, cette étape est plus complexe. De plus, deux classes de modèles sont présentes : un modèle lorsque moins de 80 % des BRAMs sont utilisées, un second modèle lorsque le nombre de BRAMs nécessaires est supérieur à environ 80 % des ressources disponibles.

5.4.3 Modélisation

Lorsque la ressource BRAM est disponible, le nombre de MLUTs est constant. Nous avons décidé donc de choisir un modèle constant pour les MLUTs. L'utilisation des registres et des LLUTs augmente avec chaque palier d'utilisation du nombre de blocs RAM. Mais cette augmentation est suffisamment faible pour ne

pas être prise en compte. Pour la modélisation des registres et LLUTs, nous avons donc choisi un nombre constant obtenu en faisant la moyenne des 10 premiers résultats de synthèse pour le FPGA Spartan6. Le nombre de blocs DSP ne varie pas suivant la largeur d'image. L'utilisation de cette ressource ne semble dépendre que du nombre de pixels en parallèle. Lorsqu'il n'y a plus de BRAMs disponible, les MLUTs sont utilisées en tant que mémoire, et l'utilisation des registres et des LLUTs évolue de la même façon.

Le paramètre largeur d'image influence principalement la taille des FIFOs présentes dans l'IP. C'est pourquoi ce sont principalement les blocs RAM qui sont impactés par une variation de ce paramètre. Le modèle attendu pour les BRAMs est une fonction du type :

$$2^{ceil(log_2(C_{IMG\_WIDTH}/C_{NB\_PIXEL}))} \tag{5.1}$$

Avec une constante multiplicative dépendant de la taille en kb des blocs RAMs dans le FPGA.

Une estimation des ressources est proposée en se basant sur les ressources utilisées par le Spartan6 dans les plages où la ressource BRAM est disponible. L'estimation des ressources (classe 1) proposée est :

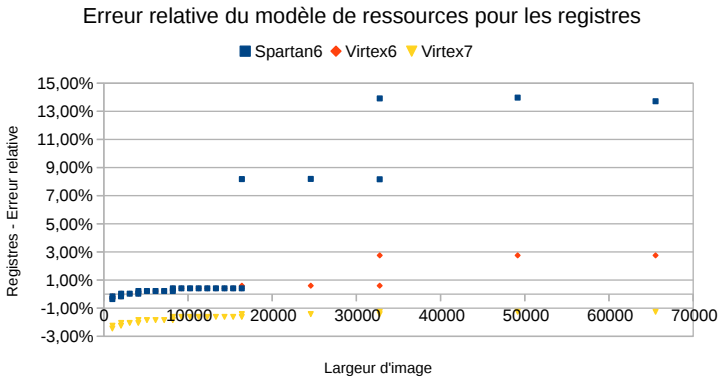
- Registres : $f(x) = 6285$
- LUTs (total) : $f(x) = 5960$
- LLUTs : $f(x) = 5946$
- MLUTs : $f(x) = 14$
- BRAMs : $f(x) = 10/128 \times 2^{ceil(log_2(x/8))}$
- DSPs : $f(x) = 8$

Ces modèles peuvent être visibles sur les figures 5.8 et 5.9.

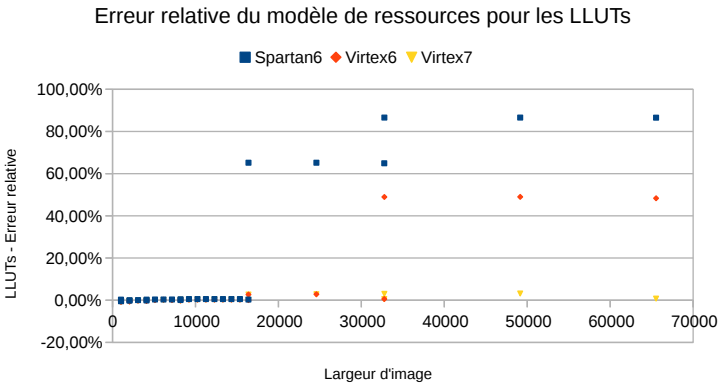
5.4.4 Validation du modèle

L'erreur relative entre les modèles et les ressources réellement utilisées est visible sur les figures 5.11 et 5.12. Nous pouvons remarquer qu'il est pertinent de supposer l'utilisation des registres et des LLUTs constant. En effet, dans les plages où l'utilisation des BRAMs n'est pas saturée, l'erreur relative est inférieure à 1 % pour les FPGAs Spartan6 et Virtex6, et inférieure à 3 % pour le Virtex7 pour ces ressources. Il en est de même pour les MLUTs. L'erreur relative pour cette ressource est égale à 0 dans les plages où l'utilisation des BRAMs n'est pas saturée. Mais cette erreur est supérieure à 99 % dès que l'utilisation des BRAMs est limitée. Comme l'utilisation des blocs RAM est plafonnée, les MLUTs sont utilisées en tant que mémoire. Cela explique l'élévation importante du nombre de MLUTs que l'on peut observer sur la figure 5.9a.

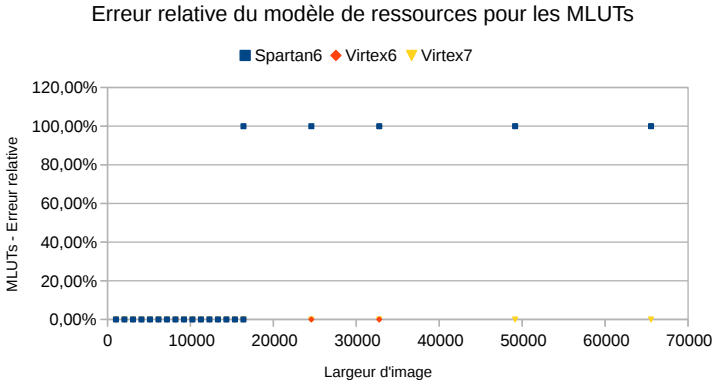
Ainsi, les deux classes de modèles peuvent être observées sur ces figures. Une première classe lorsque moins de 80 % des BRAMs sont utilisées. Une seconde classe lorsque le nombre de BRAMs nécessaire est supérieur à environ 80 % des ressources disponibles. Cette valeur est atteinte lorsque la largeur d'image est supérieure à



(a) Registres

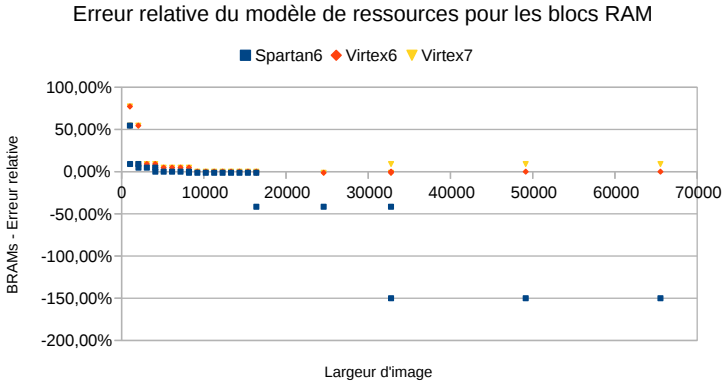


(b) LLUTs



(c) MLUTs

FIGURE 5.11 – Erreur relative entre le modèle d'utilisation des ressources selon la largeur d'image et les valeurs réelles pour les trois familles de FPGA.



(a) Blocs RAM

FIGURE 5.12 – Erreur relative entre le modèle d'utilisation des ressources selon la largeur d'image et les valeurs réelles pour les trois familles de FPGA (suite).

16384 pour le Spartan6, 32768 pour le Virtex6, et 65536 pour le Virtex7.

Nous pouvons remarquer que les modèles définis avec les données du Spartan6 sont valables pour les FPGAs Virtex 6 et 7.

5.4.5 Conclusion

L'utilisation des ressources selon la largeur d'image a été étudiée et des modèles de dimensionnement ont été extraits. Ces modèles sont valables pour toutes les familles de FPGAs étudiées tant que le taux d'utilisation des blocs RAM est inférieure à 90 %. Comme pour le nombre de pixels en parallèle, les ressources les plus critiques ici sont les blocs RAM et les MLUTs.

En effet, le nombre de blocs RAM utilisés augmente avec la largeur d'image jusqu'à ce qu'il n'y ait plus de blocs RAM disponibles. L'utilisation de cette ressource est plafonnée à une valeur comprise entre 90 et 95 % du nombre maximum de blocs RAM pour toutes les familles de FPGA (y compris le Spartan6). Pour les petites largeurs d'image, il y a un nombre minimum de blocs RAM utilisés selon la famille de FPGAs choisie. Les FGPAs de la famille Virtex utilisent deux fois moins de blocs RAM lorsque l'on augmente la largeur d'image. Cela est dû au fait que leurs blocs RAM ont une taille de 36 kb quand les blocs RAM du Spartan 6 ont une taille de 18 kb.

Pour ce paramètre, il est important de s'assurer de toujours utiliser les blocs RAM et non MLUTs lors de la synthèse. En effet, en cas de plafonnement du nombre de blocs RAM, l'utilisation des MLUTs augmente très rapidement. L'épuisement des ressources du FPGA peut donc arriver très rapidement en cas d'utilisation des MLUTs. En pratique, les capteurs matriciels les plus larges observés ont des largeurs d'image autour de 5120 pixels (On Semi PYTHON25K). Les capteurs avec des largeurs d'images plus importantes, par exemple le CMOSIS CHR70M, ne répondent à nos besoins de performances temporelles. Ainsi, il est très pro-

TABLE 5.10 – Ressources utilisées et fréquence maximale pour six valeurs du paramètre hauteur d'image

Val. Paramètre	96	128	1020	1088	2048	2050
Registres	5030	5030	6167	6570	6570	6985
LUTs (total)	4455	4455	5925	6458	6458	7049
LLUTs	4441	4441	5911	6444	6444	7035
MLUTs	14	14	14	14	14	14
BRAMs	18	18	21	22	22	22
DSP48	8	8	8	8	8	8
Période min. (ns)	6,940	6,940	7,431	7,469	7,469	7,336
Fréq. max. (MHz)	144,085	144,085	134,573	133,888	133,888	136,313

bable que dans la majorité des cas, les modèles de classe 1 soient les seuls modèles nécessaires pour ce paramètre.

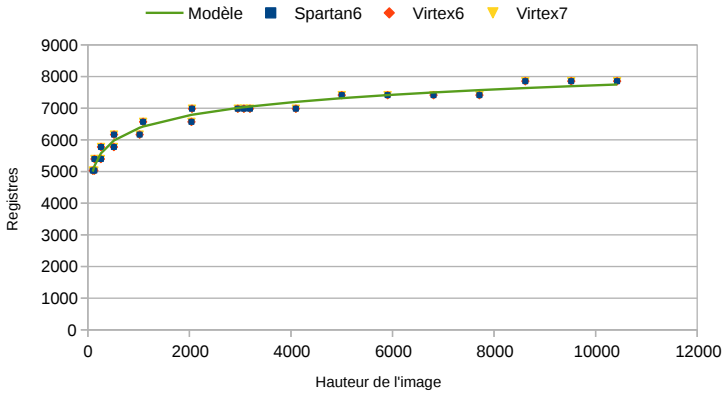
En termes de fréquence maximale, il n'y pas de changement après synthèse. Cependant, les fréquences d'horloge indiquées après synthèse sont des estimations qui ne correspondent pas toujours à la valeur réelle. Cette valeur dépend du placement-routage des opérateurs.

5.5 Hauteur d'image

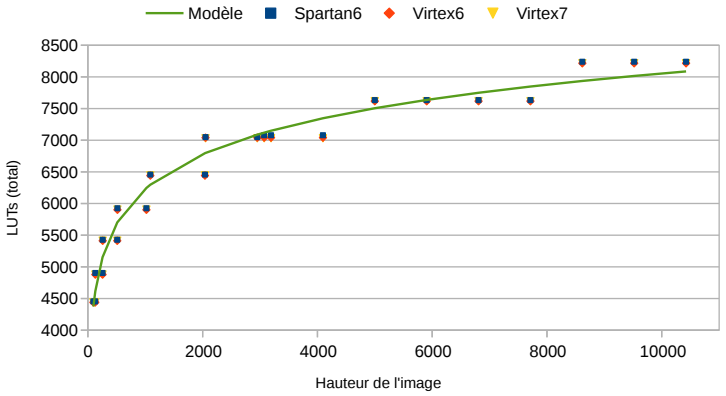
Le paramètre « hauteur d'image » influence le nombre de bits des signaux $ISUM$ et $IYSUM$ présentés dans la partie 4.4. Étant donné que le calcul de ces sommes est important pour la détection de trait laser, ce paramètre nous a semblé intéressant à étudier.

5.5.1 Collection des données

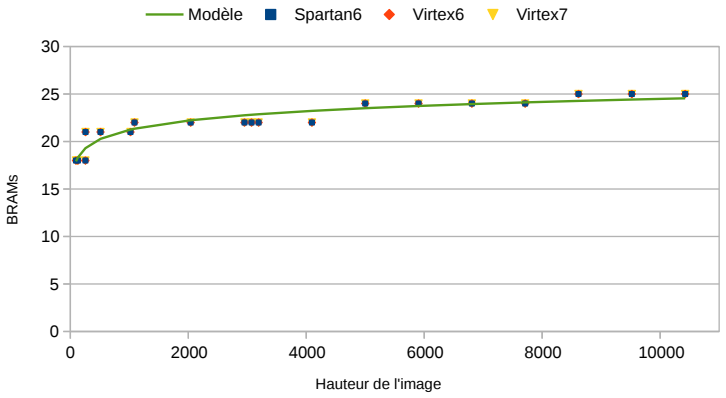
L'IP a été synthétisée pour 25 valeurs pour la hauteur de l'image entre 96 et 10424. Nous avons choisi ces valeurs afin d'avoir un nombre de paliers permettant de vérifier le modèle. Les valeurs des autres paramètres sont fixées à leur valeur par défaut indiquée dans le tableau 5.1. Les ressources utilisées par l'IP de détection de trait laser pour six hauteurs d'image sont visibles dans le tableau 5.10. L'utilisation des MLUTs et des blocs DSP en fonction de la hauteur d'image est constante. Les résultats de ces synthèses pour les registres, LUTs, et BRAMs sont disponibles sur la figure 5.13.



(a) Registres



(b) LUTs



(c) Blocs RAM

FIGURE 5.13 – Utilisation des ressources selon la hauteur d'image pour trois familles de FPGA et modèles associés obtenus partie 5.5.3 avec les données du Spartan6.

TABLE 5.11 – Paramètres associés aux données V1 à V5 pour l'étude du paramètre « hauteur d'image ».

V1	V2	V3	V4	V5
C_{IMG_HEIGHT}	Registres	LUTs (total)	LLUTs	BRAMs

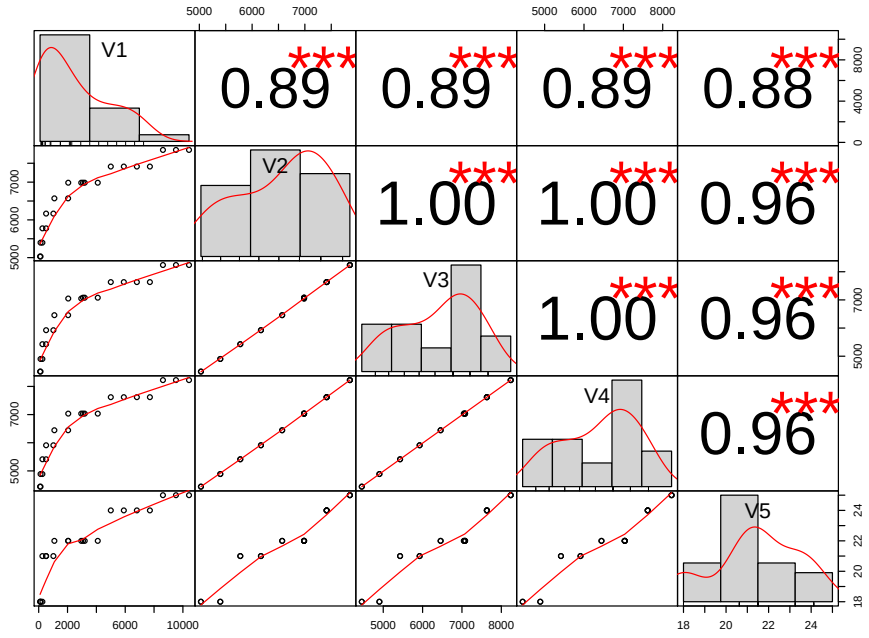


FIGURE 5.14 – Corrélations entre les valeurs du paramètre et les ressources utilisées pour le FPGA Spartan6. Figure obtenue avec le logiciel R pour le paramètre « hauteur d'image ».

5.5.2 Analyse des données

Les données collectées en fonction de la hauteur d'image ont été analysées avec le logiciel R. Les ressources analysées sont les registres, les LUTs, les LLUTs, et les BRAMs, les autres ressources étant constantes. Les données analysées sont listées dans le tableau 5.11. Les corrélations obtenues entre les ressources sont représentées sur la figure 5.14 et le tableau 5.12.

Les coefficients de corrélation obtenus sont compris entre $0,88$ et 1. Les données et ressources sont donc fortement corrélées entre elles. De plus, nous pouvons observer qu'il n'y a pas de limite de linéarité pour ce paramètre. Ainsi, des modèles mathématiques peuvent être extraits pour l'estimation des ressources selon ce paramètre. Contrairement aux deux paramètres précédents, il n'y a qu'une seule classe de modèles.

TABLE 5.12 – Corrélations entre les données V1 à V5 pour le FGPA Spartan6 pour le paramètre « hauteur d'image ».

	V1	V2	V3	V4	V5
V1	1.0000000	0.8870947	0.8908303	0.8920017	0.8769961
V2	0.8870947	1.0000000	0.9998531	0.9998800	0.9647217
V3	0.8908303	0.9998531	1.0000000	0.9999645	0.9638677
V4	0.8920017	0.9998800	0.9999645	1.0000000	0.9649586
V5	0.8769961	0.9647217	0.9638677	0.9649586	1.0000000

5.5.3 Modélisation

Sur la figure 5.13, l'évolution des registres, LLUTs et blocs RAM se fait par paliers. Ceci est dû au fait que la hauteur d'image influence la taille de signaux internes. Les fonctions en jeu pour l'utilisation des registres et des LLUTs ont approximativement la même forme que la fonction en jeu pour l'estimation des blocs RAMs selon la largeur d'image :

$$2^{ceil(log_2(C_{IMG\_HEIGHT}))} \tag{5.2}$$

Pour la ressource BRAM, même si son évolution semble suivre cette équation, il est possible que la fonction qui définit son utilisation soit plus complexe. Cette ressource est déjà dépendante des deux paramètres précédents.

Ces observations nous poussent à faire une modélisation logarithmique. Les modèles pour ces ressources ont été obtenus en faisant une régression linéaire entre les ressources utilisées par le FPGA Spartan6 et le logarithme en base 2 du paramètre. Les modèles mathématiques obtenus sont les suivants :

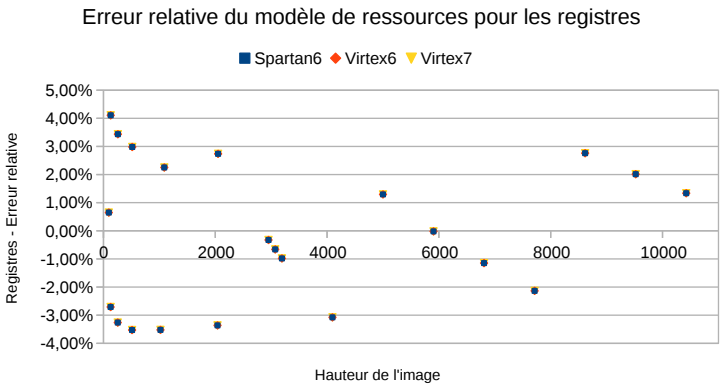
- Registres : $f(x) = 406,7406 \times log_2(x) + 2319,0676$
- LUTs (total) : $f(x) = 548,7291 \times log_2(x) + 761,9674$
- LLUTs : $f(x) = 547,7477 \times log_2(x) + 754,2951$
- MLUTs : $f(x) = 14$
- BRAMs : $f(x) = 0,9808 \times log_2(x) + 11,4514$
- DSPs : $f(x) = 8$

Ces modèles sont visibles sur la figure 5.13 avec les données collectées.

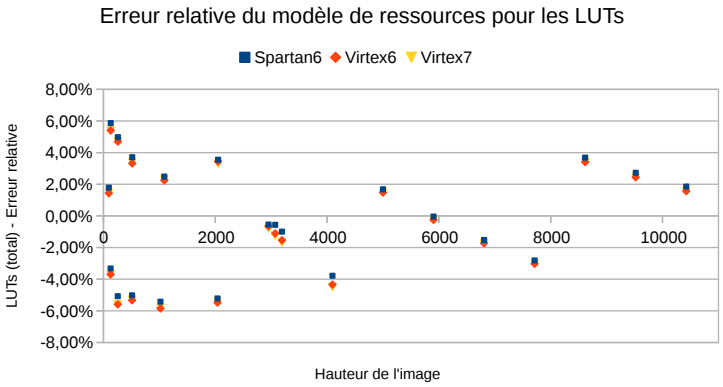
5.5.4 Validation du modèle

L'erreur relative entre les modèles mathématiques proposés et les ressources réellement utilisées a été calculée. Les résultats pour les registres, les LUTs et les blocs RAM sont présentées sur la figure 5.15.

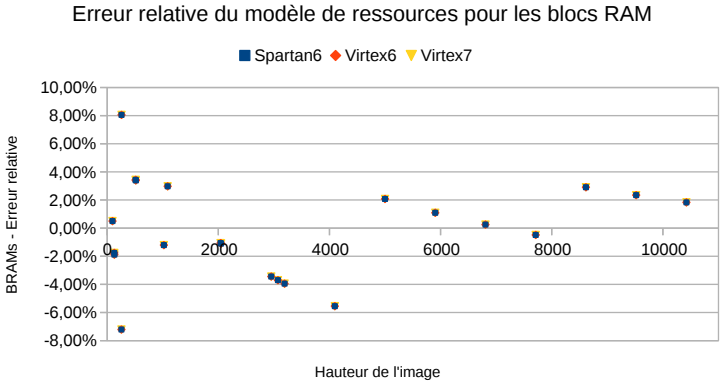
L'erreur relative est de ± 4 % pour les registres, ± 6 % pour les LUTs et de ± 8 % pour les blocs RAM. L'erreur relative est de 0 % pour les MLUTs et les blocs DSP. Les modèles mathématiques proposés permettent une estimation précise des



(a) Registres



(b) LUTs



(c) Blocs RAM

FIGURE 5.15 – Erreur relative entre le modèle d'utilisation des ressources selon la hauteur d'image et les valeurs réelles pour les trois familles de FPGA.

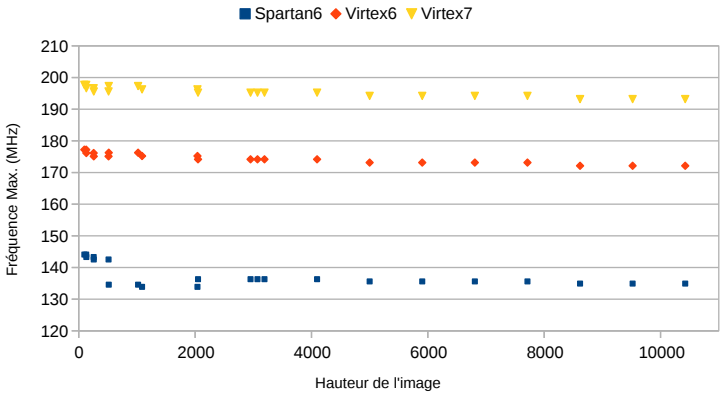


FIGURE 5.16 – Fréquence maximale après synthèse selon la hauteur d'image pour trois familles de FPGA.

ressources réellement utilisées.

5.5.5 Conclusion

Des modèles mathématiques ont été extraits selon la hauteur de l'image. L'utilisation des MLUTs et des blocs DSP est constante selon ce paramètre. L'utilisation des registres, LUTs, LLUTs et BRAMs a une évolution logarithmique selon la ressource. Cependant, le taux d'utilisation de chacune de ces ressources est inférieure à 10 %, ce qui fait que l'influence de ce paramètre n'est pas critique sur l'utilisation des ressources du FPGA. C'est pourquoi seule une classe de modèle est définie.

La fréquence maximale après synthèse est présentée sur la figure 5.16. Par rapport aux paramètres précédents, la hauteur a une influence sur la fréquence maximale de l'IP. En effet, le chemin critique de cette IP se situe dans le calcul des sommes $ISUM$ et $IYSUM$ présentées partie 4.4. La hauteur de l'image influence le nombre de bits de ces signaux et donc la taille des additionneurs et multiplieurs pour le calcul des sommes sur FPGA. Ainsi, plus la hauteur d'image est importante, plus les opérations d'addition et de multiplication seront longues. Cependant, cette diminution de la fréquence d'horloge selon la hauteur d'image est relativement faible (de l'ordre de 10 %).

5.6 Autres paramètres

Nous avons présenté l'utilisation des ressources suivant trois paramètres qui sont le nombre de pixels en parallèle (C_{NB_PIXEL}), la largeur d'image (C_{IMG_WIDTH}) et la hauteur d'image ($C_{MAX\_IMG\_HEIGHT}$). Les autres paramètres génériques de l'opérateur de détection de trait laser sont la profondeur des pixels (C_{PIX_SIZE}), le nombre de bits du résultat à transmettre au PC ($C_{LP\_POS\_SIZE}$) et la valeur par défaut du seuil de détection du trait laser ($C_{THRESHOLD}$).

Ces trois derniers paramètres ont une plage de variation inférieure aux paramètres étudiés précédemment. Il y a peu de chances d'atteindre les limites en termes de ressources avec ces paramètres. Nous avons donc décidé de n'effectuer les synthèses que pour un seul FPGA qui est le Spartan6.

5.6.1 Profondeur des pixels

La profondeur des pixels correspond au nombre de bits sur lequel sont codés les pixels. La profondeur des pixels varie habituellement de 8 à 12, voire 14 bits. Cependant, il est possible d'élever la valeur de ce paramètre jusqu'à 32.

5.6.1.1 Collection des données

Nous avons effectué des synthèses pour 13 valeurs de la profondeur des pixels comprises entre 8 et 32. Les autres paramètres génériques sont constants. Les courbes d'utilisation des ressources obtenues sont présentées sur la figure 5.17 avec les modèles obtenus partie 5.6.1.2.

5.6.1.2 Modélisation

L'évolution de l'utilisation des ressources selon le paramètre semble être linéaire pour toutes les ressources sauf pour les blocs DSP. L'utilisation des MLUTs est constante. Nous proposons donc des modèles linéaires pour ces ressources :

- Registres : $f(x) = 333,9725x + 3143,3187$
- LUTs (total) : $f(x) = 394,7308x + 2389,1538$
- LLUTs : $f(x) = 394,7308x + 2375,1538$
- MLUTs : $f(x) = 14$
- BRAMs : $f(x) = 0,6648x + 15,2418$

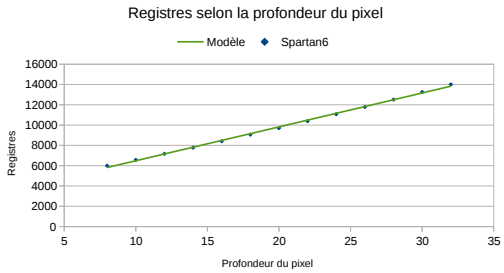
Ces modèles sont également représentés sur la figure 5.17.

5.6.1.3 Validation du modèle

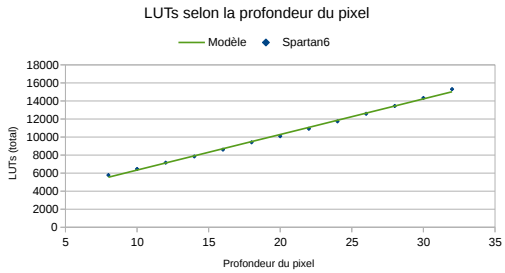
Les erreurs relatives entre les modèles proposés et les ressources réellement utilisées sont présentées sur la figure 5.18. D'après les figures obtenues, le modèle linéaire ne soit pas le meilleur modèle. En effet, les figures 5.18a et 5.18b, le modèle le plus approprié serait un polynôme d'ordre plus élevé. Cependant, les erreurs relatives pour ces deux ressources sont faibles : $\pm$ 3 % pour les registres, $\pm$ 4 % pour les LUTs et $\pm$ 4 % pour les BRAMs. Cette approximation reste donc pertinente.

5.6.2 Taille de la sortie

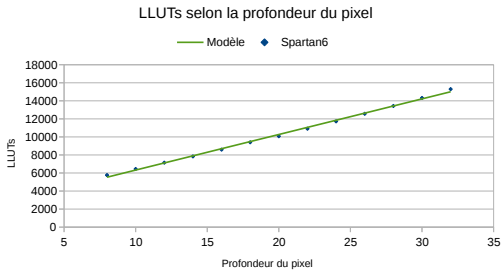
Ce paramètre correspond au nombre de bits du résultat de la détection de trait laser qu'il faut transmettre au PC hôte. Les tailles de variables sur CPU sont en général : 8 bits, 16 bits, 32 bits ou 64 bits. Au vu du faible nombre de



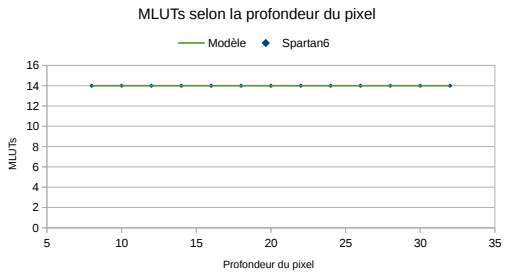
(a) Registres



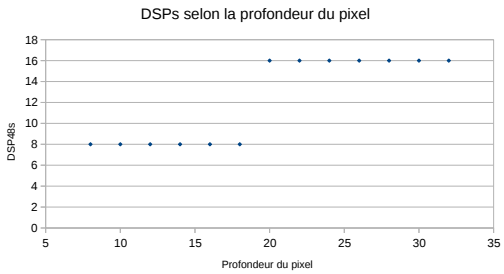
(b) LUTs



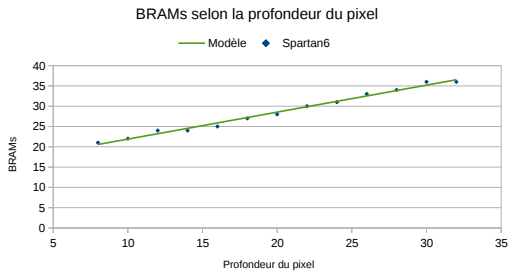
(c) LLUTs



(d) MLUTs

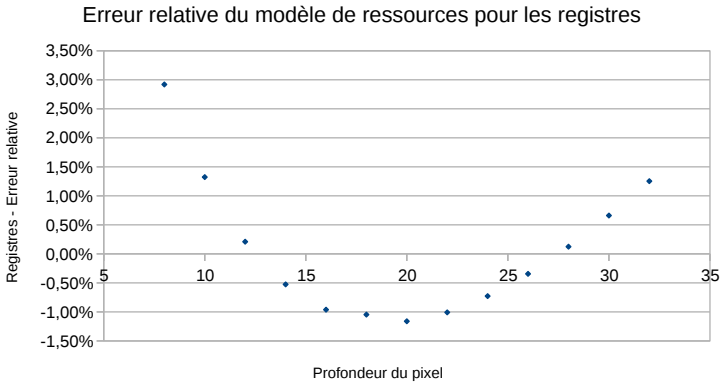


(e) Blocs DSP

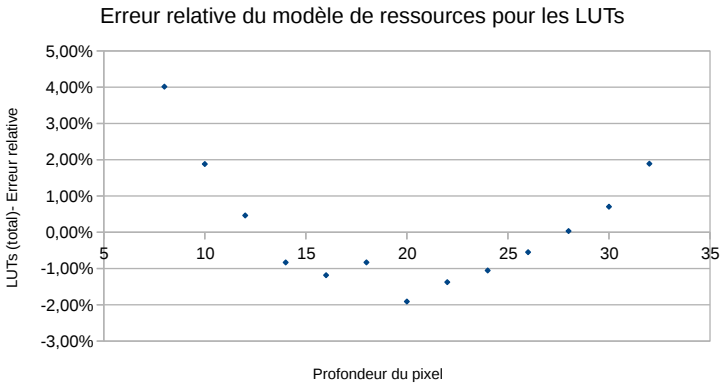


(f) Blocs RAM

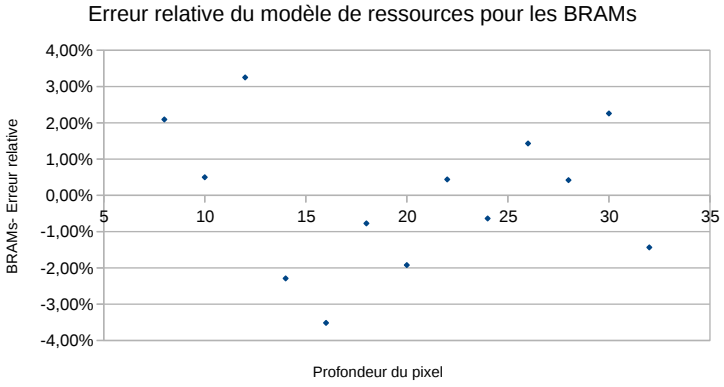
FIGURE 5.17 – Utilisation des ressources selon la profondeur des pixels pour un FPGA Spartan6 et modèles associés obtenus partie 5.6.1.2 avec les données du Spartan6.



(a) Registres



(b) LUTs



(c) Blocs RAM

FIGURE 5.18 – Erreur relative entre le modèle d'utilisation des ressources selon la profondeur des pixels et les valeurs réelles pour un FPGA Spartan6.

TABLE 5.13 – Ressources utilisées et fréquence maximale pour trois valeurs du paramètre $C_{LP\_POS\_SIZE}$

Val. Paramètre	8	16	32
Registres	6290	6570	7130
LUTs (total)	6434	6458	6506
LLUTs	6428	6444	6476
MLUTs	6	14	30
BRAMs	21	22	24
DSP48	8	8	8
Période min. (ns)	7,469	7,469	7,469
Fréq. max. (MHz)	133,888	133,888	133,888

TABLE 5.14 – Ressources utilisées et fréquence maximale pour trois valeurs du paramètre $C_{THRESHOLD}$

Val. Paramètre	40	128	240
Registres	6570	6570	6570
LUTs (total)	6458	6458	6458
LLUTs	6444	6444	6444
MLUTs	14	14	14
BRAMs	22	22	22
DSP48	8	8	8
Période min. (ns)	7,469	7,469	7,469
Fréq. max. (MHz)	133,888	133,888	133,888

valeurs possibles, il est inutile d'essayer d'extraire un modèle mathématique pour ce paramètre.

Le tableau 5.13 représente l'utilisation des ressources et la fréquence maximale de l'IP pour trois valeurs de ce paramètre. La synthèse ne peut être effectuée avec la valeur 64 bits pour le paramètre car, dans la version actuelle de l'opérateur, le résultat de la détection de trait laser a au maximum 40 bits.

5.6.3 Seuil par défaut

Comme on peut le voir dans le tableau 5.14, une modification de la valeur du seuil par défaut ne produit aucune modification au niveau des ressources utilisées. En effet, dans l'IP, ce paramètre est uniquement la valeur d'initialisation d'un signal interne qui peut être modifié par la suite par le processeur.

5.7 Conclusion

Dans ce chapitre, nous avons introduit le concept de dimensionnement des ressources. Le dimensionnement consiste à estimer des modèles d'évolution de l'utilisation des ressources selon des paramètres prédéfinis.

Les ressources de l'IP de détection de trait laser ont été dimensionnées par rapport à ses multiples paramètres. La méthodologie utilisée pour le dimensionnement a été présentée dans ce chapitre. Cette méthode est relativement efficace pour estimer les ressources utilisées par l'IP de détection de trait laser.

Les modèles ont été estimés pour un FPGA mais validés pour trois familles de FPGAs avec le même outil de synthèse et le même environnement pour la première classe de modèle. Suivant l'évolution des paramètres, les ressources peuvent avoir un comportement constant, linéaire, voire logarithmique. Cependant le taux d'occupation de ces ressources doit être pris en compte lors de la modélisation. En effet, lorsque le taux d'occupation des ressources est élevé, les modèles de classe 1 ne sont plus respectés. Dans ces conditions là, d'autres modèles doivent donc être définis. Ces modèles seront définis en tant que modèles de classe 2, 3 voire plus selon le paramètre en question. Pour cette IP, le paramètre C_{NB_PIXEL} est celui sur lequel il faut porter le plus d'attention. En effet, en plus de contenir quatre classes de modèles, il est celui pour lequel la probabilité de tomber dans la deuxième classe de modèle est la plus élevée. Le second paramètre envers lequel porter attention est la largeur d'image C_{IMG_WIDTH}. Même s'il est actuellement peu probable de tomber dans la deuxième classe de modèles, il suffit qu'un grand nombre de ressources BRAMs utilisées par le système pour passer dans cette classe.

Ces modélisations peuvent être utilisées avec de nouveaux FPGAs ou de nouveaux outils de synthèse. En effet, en connaissant l'évolution des ressources selon un paramètre, les utilisateurs peuvent avec deux synthèses seulement estimer le modèle mathématique correspondant. De même, si les utilisateurs souhaitent utiliser un autre algorithme, la méthodologie présentée peut être appliquée à nouveau afin de définir les modèles d'utilisation des ressources. Cependant, il est important de prendre en compte le taux d'occupation des ressources car les modèles ne sont valables que dans une classe définie par le pourcentage de ressources disponibles.

Chapitre 6 : Conclusion générale et perspectives

Sommaire

6.1 Conclusion

Cette thèse a été réalisée dans le cadre d'une CIFRE avec l'entreprise Pattyn Bakery Division et le laboratoire Hubert Curien. Le contexte industriel de cette thèse a été présenté dans l'introduction. L'objectif de cette thèse était de mettre en place un système de vision flexible et adaptable sur un système caméra-FPGA pour une application de triangulation laser.

Les méthodes d'acquisition tridimensionnelles passives et actives ont été présentées. La triangulation laser a été décrite en détails, ainsi que les algorithmes de détection de trait laser. La première contribution de ce travail est l'analyse et la comparaison des algorithmes de détection de trait laser présentée dans le chapitre 2. Nous en avons déduit que l'algorithme basé sur le filtrage FIR par une dérivée de gaussienne était actuellement l'algorithme le plus précis. Nous en avons également déduit que l'algorithme exploitant le centre de masse (actuellement utilisé par l'entreprise) était un bon compromis vitesse-précision, ce qui fait que son choix est encore pertinent de nos jours.

La mise en place des systèmes de vision performants se fait via l'exploitation des capacités des FPGAs. En effet, les FPGAs permettent d'exploiter différents types de parallélismes. Augmenter le nombre de données traitées par cycle d'horloge (parallélisme de données) est une méthode souvent utilisée pour traiter des débits de données importants. De plus, la technologie FPGA permet d'avoir un contrôle précis des opérations effectuées par rapport à l'horloge du système. Des caractéristiques des systèmes de vision sur FPGA ont été introduits dans le chapitre 3. Notamment, les interfaces d'acquisition et de formatage d'images, la gestion des mémoires externes, la transmission de l'image ou des résultats vers d'autres plate-formes et les types de contrôle existants.

Une architecture générique a été développée pour l'algorithme du calcul du centre de masse. Cette architecture est présentée dans le chapitre 4. Cette architecture matérielle a été implantée dans une caméra-FPGA standard du marché. L'ensemble a été validé sur un banc de test avant d'être validé sur une machine de test. Nous avons cherché à atteindre la vitesse d'acquisition maximale du capteur. Actuellement, nous avons été capable de traiter 2530 images de résolution 2048x128 pixels par seconde. Ce système peut traiter un débit de données quatre fois plus élevé que le système initial de l'entreprise.

L'utilisation des ressources selon les paramètres de l'opérateur de détection de trait laser a été évaluée. Les résultats obtenus sont présentés dans le chapitre 5. Ces informations permettront à l'entreprise de dimensionner leur système de vision selon leur besoins futurs, en particulier en ce qui concerne le choix du FPGA ou du capteur.

6.2 Perspectives

6.2.1 Industrialisation

La première perspective de cette thèse est la qualification et l'industrialisation du prototype mis en place durant la thèse. En effet, la finalité de cette thèse était de lever les limitations en termes de performances dues à l'ancien système de vision. Maintenant que ces limites sont levées, il est possible pour l'entreprise de produire des machines pouvant contrôler un nombre de produits plus important. La qualification du système de vision et son industrialisation apporteraient une plus-value importante à l'entreprise.

6.2.2 Tester d'autres algorithmes

Lors de cette thèse, l'algorithme développé et évalué était le centre de masse. Cependant, d'autres algorithmes ont été présentés dans le chapitre 2. Parmi ces algorithmes, celui qui a le plus retenu notre attention était le filtrage FIR par une dérivée de gaussienne. Cet algorithme est le plus précis mais également le plus coûteux sur CPU. Cependant, les opérations de cet algorithme sont du même type que les opérations de l'algorithme du calcul du centre de masse. Les filtres FIR sont très simples à mettre en place sur FPGA en enchaînant des registres et des blocs DSP. Utiliser la caméra-FPGA pour implanter cet algorithme semble une perspective pertinente.

6.2.3 Portage sur d'autres caméras-FPGA

Une dernière perspective intéressante est le portage des opérateurs développés sur d'autres modèles de caméra-FPGA. Des essais avaient été réalisés lors de la thèse. Cependant, la courte durée du prêt de la caméra, et le peu de documentation fourni ne nous a pas permis d'avoir les résultats escomptés. Vérifier la portabilité des IPs développées sur d'autres caméras-FPGA permettrait de prouver la généricité des opérateurs. Dans tous les cas, cette étude permettrait d'améliorer la portabilité des opérateurs.

RÉFÉRENCES

[alta] HiSPi Imager Connectivity Design Example. https://www.altera.com/support/support-resources/design-examples/intellectual-property/exm-hispi.html. Accessed : 2018-02-12.

[altb] Intel FPGA and SoC. https://www.altera.com/. Accessed : 2018-02-13.

[ALTc] ALTERA. Digital Signal Processing Blocks in Stratix Series FPGAs.

[alt13] AN 479 : Design Guidelines for Implementing LVDS Interfaces in Cyclone Series Devices. https://www.altera.com/en_US/pdfs/literature/an/an479.pdf, July 2013.

[Alt17] Altera. Video and image processing suite user guide, 2017.

[Bas] Basler. Basler time-of-flight camera | basler. https://www.baslerweb.com/en/products/cameras/3d-cameras/time-of-flight-camera/. [Online ; accessed 2017-11-30].

[BR86] François Blais and Marc Rioux. Real-time numerical peak detector. *Signal Processing*, 11(2) :145–155, 1986.

[BSF13] Saulo Vinicius Ferreira Barreto, Remy Eskinazi Sant'Anna, and Marcilio Andre Felix Feitosa. A method for image processing and distance measuring based on laser distance triangulation. In *Electronics, Circuits, and Systems (ICECS), 2013 IEEE 20th International Conference on*, pages 695–698. IEEE, 2013.

[CHT04] Pierre Cubaud, Jean-François Haas, and Alexandre Topol. Numérisation 3d de documents par photogrammétrie. In *Conférence Internationale Francophone sur l'Ecrit et le Document (CIFED 04)*, 2004.

[CL95] Brian Curless and Marc Levoy. Better optical triangulation through spacetime analysis. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 987–994. IEEE, 1995.

[Cor03] Aptina Imaging Corporation, editor. *AR0134 Developer Guide.* 2003.

[Cor13] Mentor Graphics Corporation. *Mentor® Verification IP Altera® Edition, AMBA AXI3^{TM}/AXI4^{TM} User Guide*, 2013.

[CTC95] James Clark, Emanuele Trucco, and H.-F. Cheung. Improving laser triangulation sensors using polarization. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 981–986. IEEE, 1995.

[Def12] Marc Defossez. Serial lvds high-speed adc interface. `https://www.xilinx.com/support/documentation/application_notes/xapp524-serial-lvds-adc-interface.pdf`, November 2012.

[Det07] Jérémie Detrey. *Arithmétiques réelles sur FPGA*. PhD thesis, École Normale Supérieure de Lyon, 2007.

[DZS14] Luka Daoud, Dawid Zydek, and Henry Selvaraj. A Survey of High Level Synthesis Languages, Tools, and Compilers for Reconfigurable High Performance Computing. In Jerzy Swiątek, Adam Grzech, Paweł Swiątek, and Jakub M. Tomczak, editors, *Advances in Systems Science*, volume 240, pages 483–492. Springer International Publishing, Cham, 2014.

[ENG] OPTO ENGINEERING. Systèmes de vision industrielle | Opto Engineering. http ://www.opto-engineering.fr/albert. [Online ; accessed 2017-11-20].

[EOT] EOTECH. EOTECH - Numérisation 3d - Systèmes - StereoScan neo. `http://www.eotech.fr/Numerisation-3D/Systemes/StereoScan-neo/Produits/t2/r1/i35`. [Online ; accessed 2017-11-02].

[FBR+10] Johannes Fürtler, Ernst Bodenstorfer, Michael Rubik, Konrad J. Mayer, Jörg Brodersen, and Christian Eckel. *High-Performance Smart Cameras*, pages 119–135. Springer US, Boston, MA, 2010.

[FCPR15] Virginie Fresse, Catherine Combes, Matthieu Payet, and Frédéric Rousseau. Methodological Framework for NoC Resources Dimensioning on FPGAs. *Procedia Computer Science*, 56 :391–396, January 2015.

[Fen10] Bob Feng. Implementing a tmds video interface in the spartan-6 fpga. `https://www.xilinx.com/support/documentation/application_notes/xapp495_S6TMDS_Video_Interface.pdf`, December 2010.

[FGIS05] João Guilherme DM França, Mário A. Gazziro, Alessandro Noriaki Ide, and José Hiroki Saito. A 3d scanning system based on laser triangulation and variable field of view. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 1, pages I–425. IEEE, 2005.

[FKN01] Robert Fisher and D K. Naidu. A comparison of algorithms for subpixel peak detection. 09 2001.

[For67] George E. Forsen. *Processing Visual Data with an Automaton Eye*. Stanford Research Institute, 1967. Google-Books-ID : QA4DSQAACAAJ.

[Fre16] Karl Freund. Amazon's Xilinx FPGA Cloud : Why This May Be A Significant Milestone. *Forbes*, 2016.

[FS02] J. Forest and J. Salvi. A review of laser scanning three-dimensional digitisers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*, volume 1, pages 73–78 vol.1, 2002.

[FSCP04] J. Forest, J. Salvi, E. Cabruja, and C. Pous. Laser stripe peak detector for 3d scanners. A FIR filter approach. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004*, volume 3, pages 646–649 Vol.3, August 2004.

[GR10] Sai Siva Gorthi and Pramod Rastogi. Fringe projection techniques : whither we are ? *Optics and lasers in engineering*, 48(IMAC-REVIEW-2009-001) :133–140, 2010.

[GSA10] Christos Georgoulas, Georgios Ch. Sirakoulis, and Ioannis Andreadis. Real-time stereo vision applications. In Ales Ude (Ed.), editor, *Robot Vision*. InTech, 2010.

[har17] Tom's hardware. Test processeurs : 79 modéles comparés. `http://www.tomshardware.fr/articles/comparatif-cpu-amd-intel,2-8-15.html`, 2017.

[Hep] Heptagon. Sensing through Light Products | Heptagon. `http://hptg.com/product/#sensing`. [Online ; accessed 2017-11-02].

[HLCH12] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Horaud. *Time of Flight Cameras : Principles, Methods and Applications*. SpringerBriefs in Computer Science. Springer, October 2012.

[HMT10] Yuan-Pao Hsu, Hsiao-Chun Miao, and Ching-Chih Tsai. FPGA implementation of a real-time image tracking system. In *Proceedings of SICE Annual Conference 2010*, pages 2878–2884, August 2010.

[HP98] K. Haug and G. Pritschow. Robust laser-stripe sensor for automated weld-seam-tracking in the shipbuilding industry. In *Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society, 1998. IECON '98*, volume 2, pages 1236–1241 vol.2, August 1998.

[HPM17] Nicole Hemsoth and Timothy Prickett Morgan. *FPGA Frontiers : New Applications in Reconfigurable Computing*. Next Platform Press, January 2017.

[IDS] Ensenso Stereo 3D Camera. `https://fr.ids-imaging.com/ensenso-stereo-3d-camera.html`. [Online ; accessed 2017-11-29].

[IFM] IFM. pmd3d - distance detection. `http://www.ifm.com/ifmmy/web/pmd3d_03.htm`. [Online ; accessed 2017-11-30].

[IN76] M. Ishii and T. Nagata. Feature extraction of three-dimensional objects and visual processing in a hand-eye system using laser tracker. *Pattern Recognition*, 8(4) :229–237, 1976.

[Ins08] Texas Instruments, editor. *LVDS Owner's Manual. Including High-Speed CML and Signal Conditioning.* 4th edition, 2008.

[ISA+14] M. Imran, K. Shahzad, N. Ahmad, M. O'Nils, N. Lawal, and B. Oelmann. Energy-Efficient SRAM FPGA-Based Wireless Vision Sensor Node : SENTIOF-CAM. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(12) :2132–2143, December 2014.

[ISIU99] M. A. G. Izquierdo, M. T. Sanchez, A. Ibañez, and L. G. Ullate. Sub-pixel measurement of 3d surfaces by laser scanning. *Sensors and Actuators A : Physical*, 76(1–3) :1–8, August 1999.

[JC81] Charles J. Jacobus and Robert T. Chien. Two New Edge Detectors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-3(5) :581–592, September 1981.

[Jol01] Emmanuel Jolys. *Réalisation d'un système de triangulation au laser dans le cadre d'applications dans le domaine de l'agroalimentaire.* PhD thesis, École de technologie supérieure, 2001.

[KB07] Michael Keating and Pierre Bricaud. *Reuse methodology manual : for system-on-a-chip designs.* Springer, New York, 3rd ed edition, 2007.

[KGC91] Takeo Kanade, Andrew Gruss, and L. Richard Carley. A very fast VLSI rangefinder. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1322–1329. IEEE, 1991.

[KSCK96] Jae Seon Kim, Young Tak Son, Hyung Suck Cho, and Kwang Il Koh. A robust visual seam tracking system for robotic arc welding. *Mechatronics*, 6(2) :141–163, March 1996.

[lat] Home - Lattice Semiconductor. `http://www.latticesemi.com/`. Accessed : 2018-02-13.

[Les16] GPU : la véritable consommation électrique de plus de 40 cartes graphiques. `http://www.lesnumeriques.com/carte-graphique/gpu-veritable-consommation-electrique-plus-40-cartes-graphiques-a2733.html`, 2016.

[LTA16] Griffin Lacey, Graham W. Taylor, and Shawki Areibi. Deep learning on fpgas : Past, present, and future. *arXiv preprint arXiv :1602.04283*, 2016.

[LWZC09] Bing Li, Jianlu Wang, Fei Zhang, and Lei Chen. Error analysis and compensation of single-beam laser triangulation measurement. In *Automation and Logistics, 2009. ICAL'09. IEEE International Conference on*, pages 1223–1227. IEEE, 2009.

[Mat13] S. Mattoccia. Stereo Vision Algorithms for FPGAs. In *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 636–641, June 2013.

[McD08] E. J. McDonald. Runtime FPGA Partial Reconfiguration. In *2008 IEEE Aerospace Conference*, pages 1–7, March 2008.

[mic] Microsemi | Semiconductor & System Solutions | Power Matters. `https://www.microsemi.com/`. Accessed : 2018-02-13.

[mip] MIPI Camera Serial Interface 2 (MIPI CSI-2). `https://www.mipi.org/specifications/csi-2`. Accessed : 2018-02-05.

[MMC14] S. Mattoccia, I. Marchio, and M. Casadio. A Compact 3d Camera Suited for Mobile and Embedded Vision Applications. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 195–196, June 2014.

[MS06] Milica Mitić and Mile Stojčev. An overview of on-chip buses. *Facta universitatis-series : Electronics and Energetics*, 19(3) :405–428, 2006.

[Mul06] J. M. Muller. *Elementary functions : algorithms and implementation*. Birkhäuser, Boston, 2nd ed edition, 2006.

[OF97] Stuart F. Obermann and Michael J. Flynn. Division algorithms and implementations. *IEEE Transactions on computers*, 46(8) :833–854, 1997.

[OIA03] Y. Oike, M. Ikeda, and K. Asada. A CMOS image sensor for high-speed active range finding using column-parallel time-domain ADC and position encoder. *IEEE Transactions on Electron Devices*, 50(1) :152–158, January 2003.

[Omr] MobileRobots Color Stereo Camera. `http://www.mobilerobots.com/Accessories/BumblebeeStereo.aspx`. [Online ; accessed 2017-11-02].

[Ope10] OpenCores. *Wishbone B4, WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores*. 2010.

[opt] Velociraptor EVO - Products - Optomotive. `http://www.optomotive.com/products/velociraptor-evo`. Accessed : 2018-02-14.

[OR06] Amos R. Ormondi and Jagath Chandana Rajapakse, editors. *FPGA implementations of neural networks*. Springer, Dordrecht, The Netherlands, 2006. OCLC : ocm61477766.

[PFS12] M. Prats, J.J. Fernandez, and P.J. Sanz. Combining template tracking and laser peak detection for 3d reconstruction and grasping in underwater environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 106–112, October 2012.

[PNM92] D. S. Pierce, T. S. Ng, and B. R. Morrison. A novel laser triangulation technique for high precision distance measurement. In *Industry Applications Society Annual Meeting, 1992., Conference Record of the 1992 IEEE*, pages 1762–1769. IEEE, 1992.

[QZZ+13] Li Qi, Yixin Zhang, Xuping Zhang, Shun Wang, and Fei Xie. Statistical behavior analysis and precision optimization for the laser stripe center detector based on Steger's algorithm. *Optics Express*, 21(11) :13442–13449, June 2013.

[RGFSDR12] Rafael Rodriguez-Gomez, Enrique J. Fernandez-Sanchez, Javier Diaz, and Eduardo Ros. FPGA Implementation for Real-Time Background Subtraction Based on Horprasert Model. *Sensors*, 12(12) :585–611, January 2012.

[Sch94] T. Schenk. Concepts and algorithms in digital photogrammetry. *ISPRS journal of photogrammetry and remote sensing*, 49(6) :2–8, 1994.

[SCL15] Qiucheng Sun, Jian Chen, and Chunjing Li. A robust method to extract a laser stripe centre based on grey level moment. *Optics and Lasers in Engineering*, 67 :122–127, April 2015.

[Shi72] Yoshiaki Shirai. Recognition of polyhedrons with a range finder. *Pattern Recognition*, 4(3) :243–250, 1972.

[Ste13] Carsten Steger. Unbiased extraction of lines with parabolic and Gaussian profiles. *Computer Vision and Image Understanding*, 117(2) :97–112, February 2013.

[SYI94] Kosuke Sato, Atsushi Yokoyama, and Seiji Inokuchi. Silicon range finder-a realtime range finding VLSI sensor. In *Custom Integrated Circuits Conference, 1994., Proceedings of the IEEE 1994*, pages 339–342. IEEE, 1994.

[sys] SystemC. `http://www.accellera.org/downloads/standards/systemc`. Accessed : 2018-02-13.

[TLLA12] Beau Tippetts, Dah Jye Lee, Kirt Lillywhite, and James Archibald. Review of stereo vision algorithms and their suitability for resource-limited systems. *Journal of Real-Time Image Processing*, 11(1) :5–25, December 2012.

[Tom09] Combien consomme réellement un cpu arm ? `http://www.tomshardware.fr/articles/arm-omap-ti,1-8560.html`, 2009.

[UMG10] Rubén Usamentiaga, Julio Molleda, and Daniel F. García. Fast and robust laser stripe extraction for 3d reconstruction in industrial environments. *Machine Vision and Applications*, 23(1) :179–196, July 2010.

[WCJ07] Jih-Huah Wu, Rong-Seng Chang, and Joe-Air Jiang. A Novel Pulse Measurement System by Using Laser Triangulation and a CMOS Image Sensor. *Sensors*, 7(12) :3366–3385, December 2007.

[xil] Xilinx - All Programmable. `https://www.xilinx.com/`. Accessed : 2018-02-13.

[Xil11] Xilinx. *AXI Reference Guide*, 2011.

[XIL16a] XILINX. *7 Series FPGAs GTX/GTH Transceivers User Guide, UG476*, 2016.

[Xil16b] Xilinx. Axi4-Stream Video IP and System Design Guide, 2016.

[Xil17] Xilinx. Total power advantage using spartan-7 FPGAs. `https://www.xilinx.com/support/documentation/white_papers/wp488-spartan7-power.pdf`, 2017.

[ZGWM14] S. Y. Zheng, L. Gui, X. N. Wang, and D. Ma. A real-time photogrammetry system based on embedded architecture. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5 :633–638, June 2014.

[ZJY+12] Liguo Zhang, Jianbin Jiao, Qixiang Ye, Zhenjun Han, and Wei Yang. Robust weld line detection with cross structured light and Hidden Markov Model. In *2012 International Conference on Mechatronics and Automation (ICMA)*, pages 1411–1416, August 2012.

[ZZZG08] Lei Zhang, Mingyang Zhao, Yuanyuan Zou, and Shiyi Gao. A new surface inspection method of TWBS based on active laser-triangulation. In *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, pages 1174–1179. IEEE, 2008.

Liste des publications

Conférences internationales

[1] **Seher Colak**, Virginie Fresse, Olivier Alata, Emmanuel Dumas. Proposition and Evaluation of a Real-Time Generic Architecture for a Laser Stripe Detection System on FPGA . In *International Conference on Design and Architectures for Signal and Image Processing (DASIP2017)*, Dresden, Germany, September 2017. ***[Publié]***

[2] **Seher Colak**, Virginie Fresse, Olivier Alata, Thomas Gautrais, Emmanuel Dumas. Comparative study of laser stripe detection algorithms for embedded real-time suitability in an industrial quality control context. In *International Conference on Mechanical, Electric and Industrial Engineering (MEIE2018)*, Hangzhou, China, May 2018. ***[Accepté]***

Conférences nationales

[3] **Seher Colak**, Emmanuel Dumas, Virginie Fresse, Olivier Alata. Vers une architecture générique temps réel d'un système de triangulation laser sur FPGA. GDR SoCSip, Nantes, Juin 2016. ***[Publié]***

[4] **Seher Colak**, Virginie Fresse, Olivier Alata, Emmanuel Dumas. Proposition et évaluation d'une architecture temps réel adaptable pour détection de trait laser sur FPGA. GDR SoCSip, Bordeaux, Juin 2017. ***[Publié]***