

**Département Architecture,  
Conception et Logiciels Embarqués**

Saclay, le 03 Septembre 2010

REF. : LIST/DACLE/2010-0437/ZEF

Implémentation et portage de l'algorithme  
SURF sur processeur embarqué

**Par**

**Zied El Feki**

**DRT/LIST/DACLE/LCE(CEA)**

Responsables CEA : Stéphane CHEVOBBE  
Stéphane GUYETANT

Responsables du M2R : Alain MERIGOT  
Christine PAREY

DRT – LIST –  
Département Architecture, Conception et Logiciels Embarqués  
CEA/SACLAY – 91191 GIF-SUR-YVETTE CEDEX  
TÉL : +33 (0)1 69 08 49 67 - FAX : +33(0)1 69 08 83 95 – E-MAIL :: [jean-rene.lequepeys@cea.fr](mailto:jean-rene.lequepeys@cea.fr)  
Etablissement public à caractère industriel et commercial  
R.C.S. PARIS B 775 685 019

Ce document est la propriété du CEA. Il ne peut être reproduit ou communiqué sans son autorisation.

IDENTIFICATION : Rapport LIST/DACLE/2010-0437/ZEF

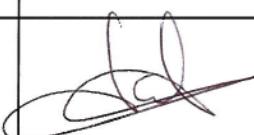
TITRE : Implémentation et portage de l'algorithme SURF sur processeur embarqué

AUTEURS : Zied EI FEKI

Mots-clefs : Détecteur, descripteur, point d'intérêt, SURF, SIFT, processeur embarqué.

UNITE : DRT/LIST/DACLE/LCE

RESUME/Contexte Ce rapport décrit les résultats d'un stage de 6 mois au cours duquel a été effectué une étude algorithmique des différents algorithmes de détection et de description de points d'intérêt dans le but de concevoir une architecture distribuée pour les détecteurs et descripteurs de points d'intérêt.

	REDACTEUR	VERIFICATEUR	CHEF DE DEPARTEMENT
NOM	Zied EI FEKI	Raphael DAVID	J.R. LEQUEPEYS
DATE	6/09/10	6/09/10	9/09/10
SIGNATURE			

## Liste de diffusion :

CEA/DRT/LIST/DACLE J.R. LEQUEPEYS

Université Paris-Sud 11 Alain MERIGOT

# Remerciements

Tout d'abord, je voudrais remercier Jean-René LEQUEPEYS, responsable du Département Architecture, Conception et Logiciels Embarqués (DACLE), ainsi que Raphael DAVID, chef du Laboratoire de Calcul Embarqué (LCE) pour m'avoir accueilli au sein du CEA LIST pour mon stage de master.

Je remercie tout particulièrement mes encadrants de stage Stéphane CHEVOBBE et Stéphane GUYETANT pour m'avoir conseillé et suivi durant tout le stage. Leurs remarques et leurs disponibilités ont été réellement bénéfiques et m'ont permis d'appréhender avec assurance mon sujet.

Je tiens également à remercier tout le personnel du LCE et tout particulièrement le personnel de la pièce 101B, avec qui j'ai beaucoup travaillé et échangé durant ce stage. L'apport de leurs diverses expériences, leur bonne humeur et leur sympathie m'ont permis d'avancer favorablement lors de ce stage.

Pour finir, je souhaite remercier toutes les personnes qui ont contribué de près ou de loin au bon déroulement de mon stage, tant d'un point de vue professionnel que personnel.

# Table des matières

<i>Introduction Générale</i> .....	<b>10</b>
<b>1. Présentation du CEA .....</b>	<b>12</b>
1.1. <i>Historique</i> .....	12
1.2. <i>Le CEA en chiffres (2009)</i> .....	12
1.3. <i>Organisation du CEA</i> .....	12
1.4. <i>Laboratoire d'Intégration des Systèmes et des Technologies (LIST)</i> .....	17
1.5. <i>Département Architecture Conception et Logiciels Embarqués (DACLE)</i> .....	18
1.6. <i>Laboratoire de Calcul Embarqué (LCE) .....</i>	19
<b>2. Contexte du stage.....</b>	<b>20</b>
2.1. <i>État de l'art des algorithmes de détection et de description de points d'intérêt .....</i>	20
2.1.1. <i>Les points d'intérêt</i> .....	20
2.1.2. <i>Les détecteurs de points d'intérêt .....</i>	21
2.1.3. <i>Présentation des principaux descripteurs de points d'intérêt .....</i>	24
2.1.4. <i>Evaluation des descripteurs.....</i>	26
2.2. <i>Conclusion .....</i>	28
<b>3. Analyse fonctionnelle et calculatoire des algorithmes SIFT et SURF .....</b>	<b>30</b>
3.1. <i>Présentation de l'algorithme SIFT.....</i>	30
3.1.1. <i>Détection des extrema dans l'espace échelle .....</i>	30
3.1.2. <i>Localisation des points d'intérêt .....</i>	33
3.1.3. <i>Affectation de l'orientation.....</i>	34
3.1.4. <i>Calcul du descripteur .....</i>	35
3.2. <i>Analyse calculatoire de SIFT .....</i>	36
3.3. <i>Bilan de l'analyse de l'algorithme SIFT .....</i>	39
3.4. <i>SURF.....</i>	39
3.4.1. <i>L'image intégrale .....</i>	40
3.4.2. <i>Le détecteur FastHessian .....</i>	41
3.4.3. <i>Construction de l'espace-échelle .....</i>	42
3.4.4. <i>Localisation des points d'intérêts .....</i>	44
3.4.5. <i>Calcul de l'orientation dominante .....</i>	45
3.4.6. <i>Calcul des composantes du descripteur .....</i>	46
3.5. <i>Analyse calculatoire de SURF .....</i>	48
3.6. <i>Bilan de l'analyse en complexité de l'algorithme SURF .....</i>	50
3.7. <i>Conclusion .....</i>	50
<b>4. Implémentation de l'algorithme SURF .....</b>	<b>52</b>
4.1. <i>Code Z-surf .....</i>	52
4.2. <i>Évaluation de Z-surf.....</i>	55
4.3. <i>Profiling de Z-surf.....</i>	59
4.4. <i>Conclusion .....</i>	61
<b>5. Portage de Z-surf sur AntX.....</b>	<b>62</b>

5.1.	<i>Le processeur AntX</i> .....	62
5.2.	<i>Adaptation de Z-surf aux contraintes matérielles</i> .....	63
5.3.	<i>Portage de la deuxième approche de Z-surf sur AntX</i> .....	66
5.4.	<i>Conception de l'interface Coprocesseur</i> .....	67
5.5.	<i>Conclusion</i> .....	71
	<b><i>Conclusion Générale</i></b> .....	72
	<b><i>Annexes</i></b> .....	74
	<b><i>Références bibliographiques</i></b> .....	84

# Table des figures

Figure 1 : Chaîne de traitement dans une caméra intelligente .....	10
Figure 2 : répartition et taille des centres civils et militaires du CEA.....	13
Figure 3 : organigramme général du CEA .....	14
Figure 4: répartition des activités au CEA-LIST .....	17
Figure 5: Constitution hiérarchique du DACLE Saclay.....	19
Figure 6 : Étapes d'analyse dans une image .....	20
Figure 7 : Différents types de points d'intérêts : les coins (a), jonctions en T (b) et point de fortes variations de texture (c).....	21
Figure 8 : Exemple d'images avant et après application de la DoG .....	22
Figure 9 : Points d'intérêt plus échelle et orientation .....	23
Figure 10 : Points d'intérêt détectés avec trois détecteurs différents.....	24
Figure 11 : Image de type Graffiti montrant la taille et l'orientation de fenêtres de descripteurs à différentes échelles .....	26
Figure 12 : Dégradation des images par flou gaussien.....	27
Figure 13 : Évaluation de 11 descripteurs par application de flou gaussien [15] .....	27
Figure 14 : Évaluation des descripteurs par application de flou pour trois types de descripteurs SIFT, CSIFT et SURF .....	28
Figure 15 : Pyramide d'une image représentée dans un espace-échelle gaussien de 4 octaves et 3 niveaux .....	31
Figure 16 : Recherche des extrema .....	32
Figure 17 : Détection des extrema dans un espace échelle gaussien constitué d'une octave à 6 niveaux .....	32
Figure 18 : Localisation des points d'intérêt.....	34
Figure 19 : Étapes de calcul de l'orientation du point d'intérêt .....	35
Figure 20 : Étapes de construction du descripteur SIFT .....	36
Figure 21 : Étapes de l'algorithme SURF .....	40
Figure 22 : Calcul de surface en utilisant l'image intégrale.....	40
Figure 23 : Approximation du Laplace des gaussiennes.....	42
Figure 24 : Les étapes d'extractions des points candidats.....	43
Figure 25 : Étapes nécessaires pour déterminer l'orientation .....	46
Figure 26 : A gauche filtre de Haar selon xx et à droite filtre de Haar selon yy.....	46
Figure 27 : Étapes nécessaires pour la construction du descripteur SURF .....	48
Figure 28 : Étapes menant à la construction du descripteur .....	50
Figure 29: Organigramme de Z-surf .....	54
Figure 30 : « Boites filtres » de taille 9x9 selon la direction horizontale et verticale.....	55
Figure 31 : Effet du passage du « float » en « int » sur la répétabilité .....	56
Figure 32 : Nombre de points d'intérêt détectés en fonction du nombre d'octaves .....	57
Figure 33 : Variation de la répétabilité en fonction du flou pour la séquence de motos.....	57
Figure 34 : Variation de la répétabilité en fonction du flou pour la séquence d'arbres .....	58
Figure 35 : Différence entre Z-surf et SURF (figure 3a motos).....	58
Figure 36 : Différence entre Z-surf et SURF (figure 3b arbres) .....	59
Figure 37 : Temps en secondes en fonction du nombre d'octaves.....	59
Figure 38 : Pipeline du processeur de contrôle constitué de 5 étages avec 16 registres généraux de 32 bits.....	62
Figure 39 : Nouvel organigramme de Z-surf .....	64

Figure 40 : Deuxième approche pour l'extraction des points d'intérêt.....	65
Figure 41 : Troisième approche pour l'extraction des points d'intérêt .....	65
Figure 42 : Quatrième approche pour l'extraction des points d'intérêt .....	66
Figure 43 : zone mémoire occupée pour chaque composante .....	66
Figure 44 : Organigramme du séquenceur .....	69
Figure 45 : Architecture du Coprocesseur.....	70
Figure 46 : « Boites filtres » de taille 9x9 selon la direction horizontale et verticale.....	78

# Liste des tableaux

Tableau 1 : Comparaison de différents détecteurs de points d'intérêt .....	24
Tableau 2 : Temps CPU moyen de SIFT, CSIFT et SURF pour la détection et la description des points d'intérêt .....	28
Tableau 3 : Valeurs des échelles à travers les octaves et les niveaux .....	43
Tableau 4 : Comparaison de SIFT et SURF .....	50
Tableau 5 : Valeurs des échelles dans SURF et Z-surf .....	53
Tableau 6 : Temps processeurs occupé de chaque fonction pour l'octave 3 cas du tri bulle ...	60
Tableau 7 : Temps processeurs occupé de chaque fonction pour l'octave cas du tri par binning .....	60
Tableau 8 : Profiling des sous-fonctions de « Fashessien » et de « Descripteur ».....	60
Tableau 9 : Résultats de profiling .....	67
Tableau 10 : Nombre de cycles nécessaires pour envoyer le résultat à AntX.....	71

# Introduction Générale

---

Le premier système de vision numérique a été réalisé en 1969, lorsque deux chercheurs des laboratoires Bell, les Américains Williard Boyle et George Smith, dessinent la structure de base d'un CCD et définissent ses principes de fonctionnement. Quelques mois plus tard, ils mettent au point la première caméra vidéo au monde fonctionnant avec un CCD.

Depuis ce jour, les systèmes de vision numériques témoignent d'un intérêt croissant, d'une part du fait des technologies meilleur marché, et d'autre part de la diversité des besoins applicatifs. En effet, ils peuvent être utilisés pour contrôler une chaîne de fabrication, asservir le déplacement d'un robot par l'intermédiaire d'une caméra qui observe l'évolution du robot et le renseigne sur sa position par rapport à la cible, la robotique mobile (évitement d'obstacles, découverte d'un monde inconnu et reconstruction d'une carte, conduite automatique et assistée)...

Aujourd'hui, les technologies d'intégrations matérielles permettent d'obtenir des systèmes de vision miniatures embarqués intégrant à la fois la partie capteur et les traitements. Or, contrairement aux appareils fixes, les systèmes embarqués sont soumis à de fortes contraintes en termes de taille et de consommation électrique. De plus, les traitements étant de plus en plus variés et complexes, les éléments de calcul doivent présenter une grande flexibilité et une puissance de calcul importante.

Une caméra intelligente est un système de vision compact qui capture des images et les interprète. Le principe est d'intégrer dans un volume réduit, toute la chaîne de traitement d'image, depuis le capteur jusqu'au processeur de calcul (figure 1). Ensuite les résultats d'exploitation peuvent être envoyés vers des organes de commandes ou de contrôles [1].

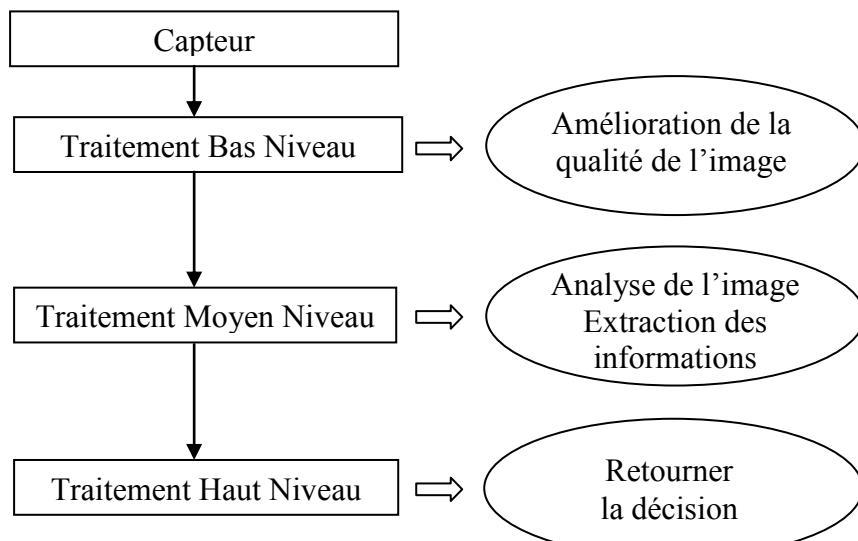


Figure 1 : Chaîne de traitement dans une caméra intelligente

Les traitements d'une chaîne de traitement d'image classique se divisent en trois niveaux : Le bas niveau, est associé aux opérations telles que la convolution, amélioration de la qualité de l'image et amélioration des contours (traitement par pixels 1000Mbps). Dans le moyen niveau, des mesures sur les objets trouvés dans l'image sont effectuées pour analyser leurs qualités et leurs propriétés dans le but de prendre des décisions par rapport aux contenus trouvés (traitement par objet de 1 à 300kilos objets). Enfin, dans le haut niveau la décision est prise et renvoyée à l'utilisateur.

Il existe beaucoup de travaux pour les traitements bas niveau ce qui n'est pas le cas du moyen niveau. C'est pourquoi nous nous sommes intéressés dans le cadre de mon stage à la partie moyen niveau : « Analyse et extraction d'informations de l'image ». L'objectif consiste à faire une étude algorithmique des techniques de détection et de description des points d'intérêt adaptées pour les systèmes embarqués. Ensuite, porter un algorithme sur processeur embarqué pour évaluer la complexité calculatoire de ce type d'algorithme. Puis, basé sur l'analyse de ce portage, proposer des pistes d'architectures matérielles embarqués optimisé pour les descripteurs des points d'intérêt.

La méthodologie suivie au cours du stage est la suivante :

- Étudier les différents algorithmes de détection et de description des points d'intérêt
- Cerner les méthodes performantes ayant la perspective d'un portage sur structure matérielle.
- Choisir l'algorithme ayant le meilleur compromis performance qualité.
- Implémenter cet algorithme et le porter sur un processeur embarqué.
- Concevoir une architecture matérielle embarqué pour la détection et la description des points d'intérêt.

Ce rapport est organisé en cinq chapitres : Le premier consiste en une présentation synthétique du CEA, ainsi que des activités du laboratoire LCE, lieu de déroulement du stage. Le deuxième chapitre comporte un état de l'art des différents détecteurs et descripteurs de points d'intérêt. Après l'étude de ces algorithmes, une analyse de performances et de la complexité calculatoire des algorithmes les plus performants est présentée dans le troisième chapitre. Un quatrième chapitre décrit les résultats et l'évaluation de l'implémentation de l'algorithme le plus performant. Un cinquième chapitre présente le résultat de portage de cet algorithme dans le processeur embarqué développé au sein du laboratoire LCE AntX. Finalement, une conclusion liste les résultats auxquels a abouti cette étude de stage de master.

# 1. Présentation du CEA

---

## 1.1. Historique

Le CEA a été créé le 18 octobre 1945 par Charles de Gaulle. C'est un organisme public de recherche scientifique français. Il intervient dans quatre grands domaines : les énergies décarbonées, la défense et la sécurité globale, les technologies pour l'information, et les technologies pour la santé.

Il est créé à la base pour poursuivre les recherches scientifiques et techniques en vue de l'utilisation du nucléaire dans les domaines de la science, de l'industrie et de la défense nationale. Originellement nommé Commissariat à l'Energie Atomique, il change d'appellation le 9 mars 2010 et devient le Commissariat à l'Energie Atomiques et aux Energies Alternatives.

Le CEA emploie environ 15 000 salariés et s'impose comme un acteur majeur de la recherche, du développement et de l'innovation sur le plan international. De plus, la richesse des partenariats régionaux et internationaux qu'il a tissés lui confère un statut particulier auprès des pouvoirs publics.

## 1.2. Le CEA en chiffres (2009)

- Plus de 15000 salariées
- 10 centres de recherche
- 51 unités mixtes de recherche (UMR)
- 120 start-up créées depuis 1984 dans le secteur des technologies innovantes
- 1 360 thésards et 289 postdocs accueillis au CEA
- 3,9 milliards d'euros de budget
- 585 dépôts de brevets prioritaires
- 350 projets européens en cours, avec la participation du CEA
- 4 079 publications en 2008 dans des revues à comité de lecture

## 1.3. Organisation du CEA

Le CEA est implanté sur dix sites en France séparés en deux catégories : les centres d'études civils et les centres d'études pour les applications militaires. Ceux-ci sont présentés dans la figure suivante :

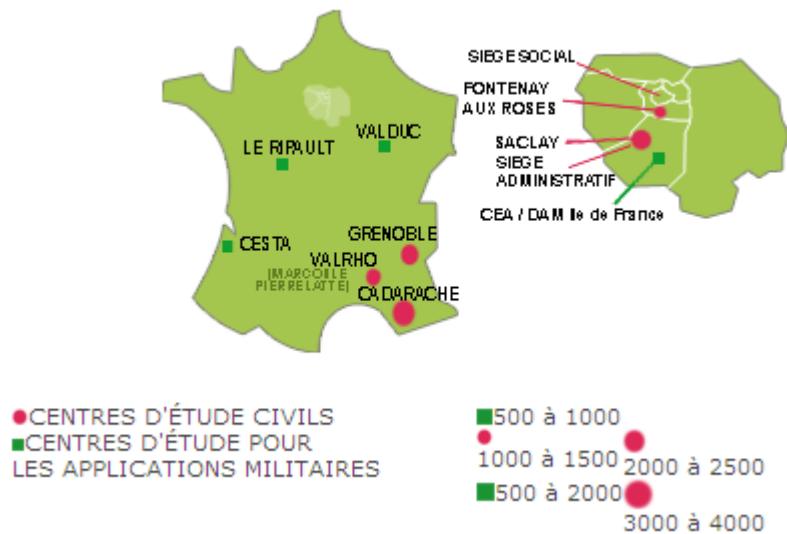


Figure 2 : répartition et taille des centres civils et militaires du CEA

Le CEA est organisé en quatre grands pôles comme on peut le voir sur la figure III.2 :

- Pôle défense ;
- Pôle nucléaire ;
- Pôle recherche fondamentale (sciences du vivant et de la matière) ;
- Pôle recherche technologique.

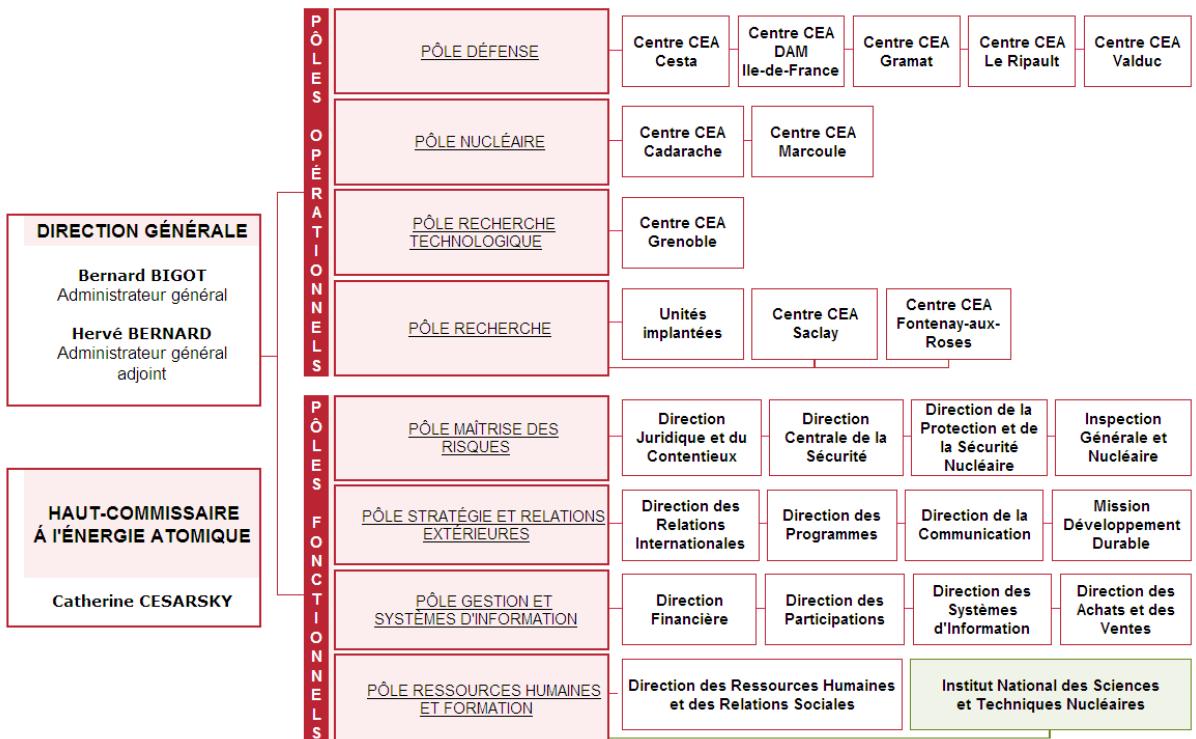


Figure 3 : organigramme général du CEA

### *Pôle nucléaire (DEN)*

En appui des industriels, le CEA cherche à optimiser le parc actuel des réacteurs nucléaires et à mettre au point des solutions techniques pour la gestion des déchets radioactifs.

Il participe aux programmes de recherches internationaux sur les réacteurs et combustibles nucléaires du futur qui assureront une production à la fois plus économique, plus sûre et générant moins de déchets.

Les recherches du CEA soutiennent également l'essor des nouvelles technologies pour l'énergie : le solaire photovoltaïque, les batteries électriques, l'hydrogène, la biomasse...

### *Pôle défense (DAM)*

La stratégie de défense française est basée sur la dissuasion nucléaire. Dans ce contexte, la Direction des Applications Militaires, qui constitue le pôle Défense du CEA, conçoit, fabrique, maintient en condition opérationnelle, puis démantèle les têtes nucléaires qui équipent les forces océaniques et aéroportées.

Le pôle Défense doit être en mesure de garantir sur le long terme la sûreté et la fiabilité des têtes nucléaires qu'il met à la disposition des armées. Après l'arrêt définitif des essais nucléaires, cette garantie devra être apportée, pendant toute la durée de vie des armes, par la simulation numérique. C'est l'objectif du programme Simulation.

La DAM est également responsable de l'approvisionnement en matières nucléaires pour les besoins de la Défense, dans le respect des décisions prises par la France d'arrêter la production de matières fissiles destinées aux armes et de démanteler les usines de production.

Elle est chargée de la conception et de l'entretien des réacteurs nucléaires assurant la propulsion des bâtiments de la Marine nationale, sous-marins et porte-avions.

Enfin, la DAM contribue, pour les instances nationales et internationales, à la surveillance du respect du Traité d'interdiction des essais nucléaires (Tice) et à la lutte contre la prolifération nucléaire et le terrorisme. Elle pilote l'ensemble des actions de recherche et développement conduites par le CEA dans le domaine de la sécurité globale.

#### *Pôle recherche fondamentale (DSM et DSV)*

La Direction des sciences de la matière (DSM) est le pôle de recherche fondamentale du CEA en physique et en chimie.

Les recherches de la DSM sont orientées sur des domaines très variés tels que ceux de l'énergie et de l'environnement (chimie, physique), des sciences de la matière pour l'innovation industrielle (nanophysique), de l'utilisation des technologies du nucléaire en recherche biologique et médicale, de la connaissance de la matière (physique des noyaux et des particules, astrophysique).

Les objectifs de la Direction des Sciences du Vivant (DSV) du CEA sont d'utiliser les méthodologies générées par le nucléaire pour développer les technologies pour la santé et, plus particulièrement, comprendre les effets sur le vivant des rayonnements et des toxiques issus des activités nucléaires.

Fondées sur la spécificité du nucléaire, les recherches dans le domaine des sciences du vivant s'inscrivent dans les trois axes stratégiques du CEA : technologies pour l'information et la santé ; énergie ; défense et sécurité globales. Elles s'attachent à ce que les technologies générées par le nucléaire bénéficient à la santé. Elles fournissent les données nécessaires pour évaluer l'impact des activités nucléaires sur notre environnement et notre santé. Enfin, elles préparent les connaissances et outils pour répondre aux demandes liées à des préoccupations de sécurité globale (maladies émergentes, bioterrorisme,...).

Le premier de ces objectifs concerne les applications des technologies générées par le nucléaire aux technologies pour la santé et aux biotechnologies. Les recherches menées visent

à développer de nouveaux outils indispensables pour appréhender la structure et le fonctionnement du vivant dans toute sa complexité. Les outils et connaissances ainsi développés sont notamment mis en œuvre dans le cadre des recherches en radiobiologie et toxicologie nucléaire qui constituent le second axe de recherche.

Le second axe de recherche privilégié porte sur la recherche biologique et médicale pour l'énergie nucléaire. Il s'agit de comprendre et d'évaluer les effets des activités nucléaires sur la santé et sur l'environnement, en particulier aux faibles doses d'exposition. L'enjeu de ces recherches est de pouvoir accéder à une évaluation scientifique et rationnelle des risques radiologiques et chimiques associés aux activités nucléaires, à court et à long termes, afin d'améliorer les méthodes de protection et d'établir des normes plus sûres, c'est-à-dire ni trop tolérantes, ni inutilement sévères donc coûteuses.

### *Pôle recherche technologique*

Installée au cœur d'un environnement scientifique, industriel et universitaire très riche, à Saclay et à Grenoble, la Direction de la Recherche Technologique consacre l'essentiel de ses recherches au développement des nouvelles technologies, dans les domaines de l'énergie, de la santé, de l'information et de la communication. Des piles à combustibles aux nanomachines, en passant par les matériaux et les biopuces, la DRT est à la pointe de la recherche technologique et participe activement au transfert de ces connaissances vers l'industrie.

Ce pôle est constitué de trois unités de recherche :

- le Laboratoire d'Electronique et des Technologies de l'Information (LETI) principalement impliqué dans l'intégration des matériaux émergeants et dans les micro et nanotechnologies au service des technologies de l'information ;
- le Laboratoire d'Innovation pour les Technologies des Energies nouvelles et les Nanomatériaux (LITEN) traite les problèmes liés aux nouvelles technologies de l'énergie (piles à combustible, hydrogène, efficacité énergétique) ;
- le Laboratoire d'Intégration des Systèmes et des Technologies (LIST) impliqué dans les technologies de l'information et de la communication.

Le laboratoire d'accueil dans lequel s'est effectué le stage se situe dans la Direction de la Recherche Technologique.

## 1.4. Laboratoire d'Intégration des Systèmes et des Technologies (LIST)

Situé en Ile de France sud (Saclay et Fontenay aux Roses), le LIST est un centre de recherche technologique sur les systèmes à logiciel prépondérant organisé selon trois thématiques présentant de forts enjeux sociétaux et économiques :

- les Systèmes Embarqués (architectures et conception de systèmes, méthodes et outils pour la sûreté des logiciels et des systèmes, systèmes de vision intelligents) ;
- les Systèmes Interactifs (ingénierie de la connaissance, robotique, réalité virtuelle et interfaces sensorielles) ;
- les Capteurs et le traitement du signal (instrumentation et métrologie des rayonnements ionisants, capteurs à fibre optique, contrôle non destructif)

Fort de la culture projet de ses 450 chercheurs, ingénieurs et techniciens, le LIST mène ses recherches en partenariat avec les grands acteurs industriels du nucléaire, de l'automobile, de l'aéronautique, de la défense et du médical pour étudier et développer des solutions innovantes adaptées à leurs besoins.

Le LIST, dans une dynamique de recherche qui va du concept de système jusqu'au démonstrateur préindustriel, contribue au transfert de technologies et favorise l'innovation notamment par l'émergence de nouvelles entreprises. Ainsi, les technologies du LIST ont suscité la création de plusieurs startups.

Les équipes du LIST sont partenaires de nombreux laboratoires universitaires, de grandes écoles et d'autres organismes de recherche au travers des projets de recherche collaborative.

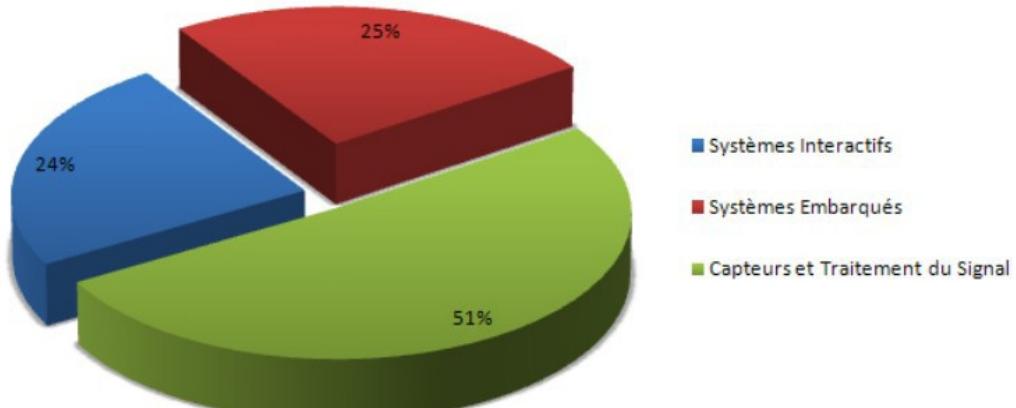


Figure 4: répartition des activités au CEA-LIST

## **1.5. Département Architecture Conception et Logiciels Embarqués (DACLE)**

Le Département Architecture, Conception et Logiciels Embarqués est un regroupement d'activités de deux laboratoires de la DRT : le LIST et le LETI (Laboratoire d'Electronique et de Technologie de l'Information basé à Grenoble). Ce regroupement, effectif depuis le 1<sup>er</sup> Janvier 2010, rassemble les forces et les compétences des deux laboratoires en termes de conception de circuits intégrés et d'architectures des logiciels pour les systèmes embarqués.

Pour renforcer cette collaboration, un laboratoire (le LIALP (Laboratoire d'Infrastructure et Atelier Logiciel pour Puce)), basé à Grenoble a été créé. Ce laboratoire est cogéré par le LIST et le LETI.

Au sein du LIST, le DACLE du CEA Saclay est chargé :

- d'étudier et de concevoir les architectures et systèmes intégrés de traitement de l'information à haut niveau de performance ainsi que les outils logiciels nécessaires à leur développement et mise en œuvre ;
- de développer des solutions pour les circuits intégrés numériques complexes, avec une emphase particulière sur les architectures massivement parallèles, avec des consommations électriques optimisées et des mécanismes de tolérances aux fautes ;
- d'étudier et concevoir des architectures pour systèmes embarqués fiables et robustes ;
- de participer activement à la construction et l'animation du Centre d'Intégration et de Conception de PILSI (Pôle International du Logiciel et Systèmes Intelligents).

Le DACLE Saclay comprend pour la partie LIST :

- l'échelon Direction (DACLE/DIR)
- le Laboratoire Calcul Embarqué (LCE)
- le Laboratoire Fiabilisation des Systèmes Embarqués (LFSE)
- le LABoratoire des fondements des Systèmes Temps Réel Embarqués (LASTRE)

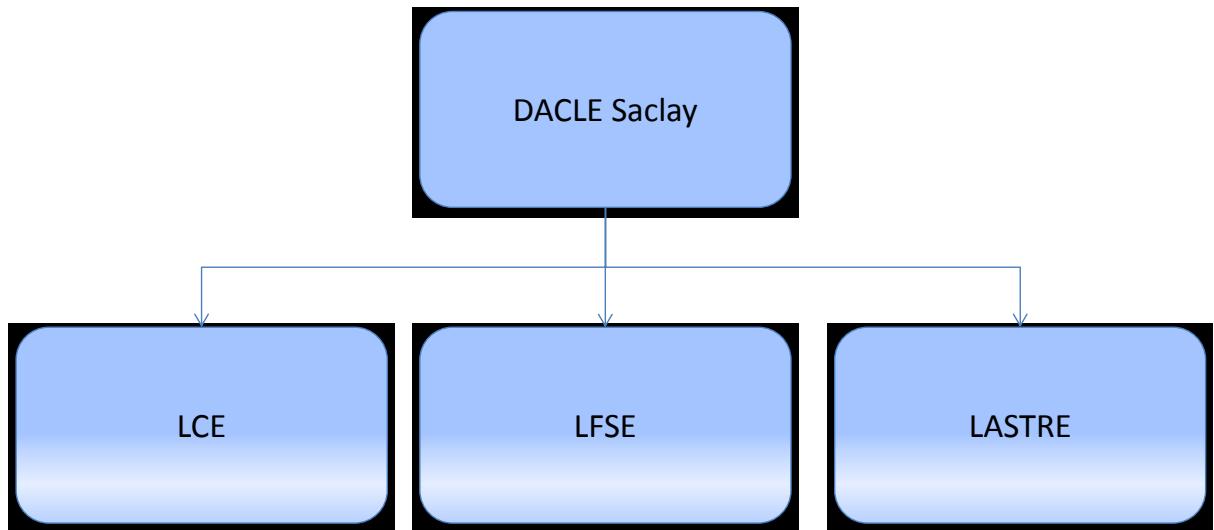


Figure 5: Constitution hiérarchique du DACLE Saclay

### 1.6. Laboratoire de Calcul Embarqué (LCE)

Le LCE est situé géographiquement sur le centre de Saclay. Il possède une grande expérience dans la conception d'architectures de calcul fortement intégrées. Ses thèmes de recherche portent principalement sur les architectures de calcul pour les SoC (System on Chip). Le LCE possède notamment une grande expérience dans les domaines des architectures de calculs parallèles, avec le développement de systèmes de contrôle matériels ; ainsi que dans le domaine des architectures reconfigurables, avec le développement de traitements matériels adaptatifs. Il développe aussi des calculateurs dédiés au traitement d'image en collaboration étroite avec le DIASI/LVIC, pour permettre une accélération matérielle des algorithmes développés et atteindre des objectifs d'embarquabilité.

## 2. Contexte du stage

---

### 2.1. État de l'art des algorithmes de détection et de description de points d'intérêt

L'analyse d'image est la reconnaissance des éléments contenus dans l'image. Son but, est de fournir une description quantitative de l'image ou une reconnaissance de forme. Elle trouve des applications dans de nombreux domaines : sciences des matériaux, sciences de la vie, géologie, robotique...

L'analyse des images par la technique des descripteurs locaux s'effectue en deux phases : détection et description des points d'intérêts (figure 6).

Ce chapitre présente un état de l'art des méthodes de détection et de description des points d'intérêt. Il est divisé en trois parties. Une première partie dans laquelle la notion de points d'intérêt est expliquée, une deuxième et troisième partie dans lesquelles sont présenté respectivement quelques détecteurs et descripteurs de points d'intérêt.

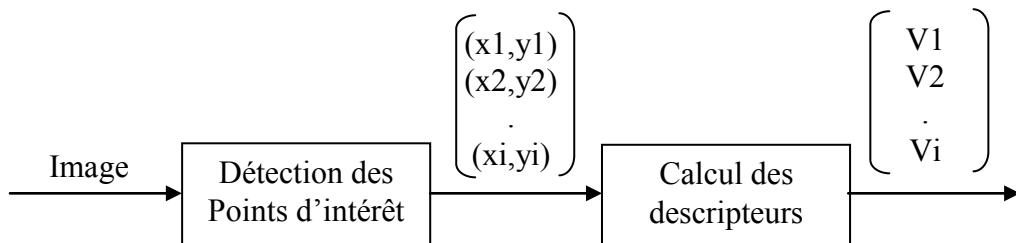


Figure 6 : Étapes d'analyse dans une image

#### 2.1.1. Les points d'intérêt

Un point d'intérêt est un point représentant un caractère spécifique de l'image, il correspond à un point de l'espace détecté de sorte à ce qu'il soit robuste aux changements de points de vue, aux changements d'éclairages et aux déformations.

Les points d'intérêt sont utilisés dans plusieurs applications telles que la reconnaissance d'objets, la détection de mouvement, le suivi, la modélisation 3D, etc.

Quelques exemples de points d'intérêt sont présentés dans la figure 7 : les coins, les jonctions en T, les points noirs sur fond blanc ou les points de fortes variations de texture.

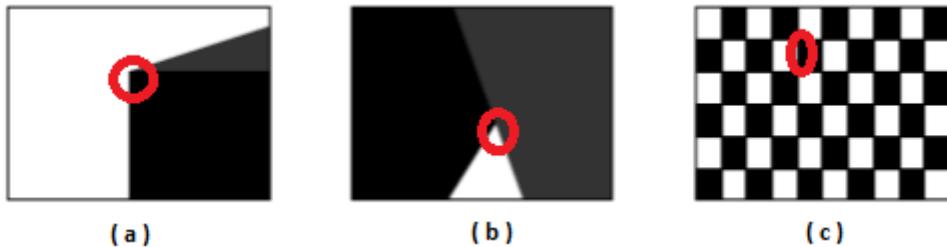


Figure 7 : Différents types de points d'intérêt : les coins (a), jonctions en T (b) et point de fortes variations de texture (c)

Un point d'intérêt est défini par :

- Une position ( $x, y$ ) bien définie dans l'image.
- Une région riche en informations autour du point.

Un bon point d'intérêt doit avoir les propriétés suivantes :

- Répétabilité : il s'agit de pouvoir détecter le même point de l'espace dans plusieurs images prises dans des conditions de vues différentes.
- Précision : les points d'intérêt détectés doivent être correctement localisés dans l'espace image tout en considérant l'échelle.
- Efficacité : les points d'intérêts détectés dans une nouvelle image doivent permettre d'effectuer des applications en temps réel.
- Robustesse : stable aux différentes déformations que peuvent subir l'image (transformations affines, rotation ou translation, changement d'échelle), aux bruits, aux artefacts de compression, au flou, etc.

### 2.1.2. Les détecteurs de points d'intérêt

Les détecteurs des points d'intérêt ont été introduits par Moravec en 1980 [2]. L'idée du détecteur de Moravec est de considérer le voisinage d'un pixel et de déterminer les changements moyens d'intensité dans le voisinage considéré lorsque le voisinage se déplace dans toutes les directions.

En 1988, Harris et Stephens [3] ont montré que le détecteur de Moravec fonctionne dans un contexte limité. Ils ont identifié certaines limitations à savoir : le caractère discret des directions de changement que l'on peut effectuer qui sont des pas de 45 degrés, la réponse du détecteur qui est bruitée et la réponse trop forte du détecteur au contour.

En corrigeant ces limitations, ils ont déduit un détecteur de coins « le détecteur de Harris ». Ce détecteur se base sur la matrice d'autocorrélation. Cette matrice décrit la distribution du gradient dans le voisinage local du point. Les points retenus dans le détecteur de Harris sont les points d'intersection de plusieurs contours ou les points d'inflexions ou les points de courbures maximales.

En [4] Smith et Brady ont introduit le détecteur de coin Smallest Unvalue Segment Assimilating Nucleus (SUSAN) qui utilise une technique différente pour détecter les points d'intérêt. Au lieu d'évaluer le gradient local qui est plus sensible au bruit et plus couteux en temps de calcul, une approche morphologique est utilisée. Cette approche consiste à considérer un cercle de rayon fixe autour du point. La valeur du centre du cercle est utilisée comme une référence. Ainsi, la décision de considérer un point comme un coin ou non dépend du nombre de pixels similaire au point central.

Cependant, les détecteurs Harris et SUSAN ne sont pas invariant au changement d'échelle. Lindeberg en 1998 [5] a introduit le concept de détection automatique d'échelle. Il a montré que chaque point possède une échelle caractéristique et que cette échelle est déterminée en utilisant l'opérateur Laplacien dans l'espace échelle. Le point d'intérêt et son échelle caractéristique définissent alors une région d'intérêt.

Ce concept a été utilisé par Mikolajczyk et Schmid en 2002 [6]. Ils ont introduit deux nouveaux détecteurs « Harris-Laplace/Affine » qui utilisent une approche multi-échelle et permettent de caractériser le voisinage du point d'intérêt. Ce sont des détecteurs, qui fonctionnent de manière itérative, la détection des points d'intérêt utilise le détecteur d'Harris.

L'idée de multi-échelle est aussi exploitée par Lowe [7] pour détecter des points en leur associant une échelle et une orientation (Figure 9). Ces détecteurs sont très performants en termes de qualité de détection mais ils sont beaucoup plus couteux en temps de calcul. Pour la recherche des points, Lowe utilise une pyramide de DoG (Difference of Gaussian), afin de détecter des objets plans qui tolèrent des déformations locales significatives (rotation, changement d'échelle...) et des variations légères de luminosité. La figure 8 montre une image à laquelle la DoG a été appliquée.



Figure 8 : Exemple d'images avant et après application de la DoG



Figure 9 : Points d'intérêt plus échelle et orientation

Dernièrement, Mikolajczyk et Schmid [8] ont proposé deux détecteurs Hessian Laplace et Hessian Affine. Ces deux détecteurs se basent sur la matrice Hessienne pour extraire un plus grand nombre de points d'intérêt qui résulte d'une bonne couverture de l'image. Ils assurent l'invariance d'échelle. De plus ils implémentent des mécanismes d'invariance par transformation affine. L'ajout de cette invariance permet d'améliorer d'une manière considérable la qualité du détecteur puisqu'elle améliore la répétabilité [9,10].

Une autre approche utilisée pour détecter les points d'intérêt est le seuillage. Cette approche a été utilisée par Matas *et al* dans leur détecteur Maximally Stable Extremal Regions (MSER) [11]. MSER permet d'effectuer un seuillage approprié à l'image. Le mot « extremal » réfère à la propriété que tous les pixels dans MSER ont une plus grande ou une plus petite intensité que les pixels se trouvant dans la frontière externe. L'expression « maximally stable » décrit la propriété optimisée dans le processus de sélection du seuil.

Récemment, en 2006, le détecteur FAST (Features from Accelerated Segment Test) a été proposé [12]. Ce détecteur considère les pixels sur un cercle de Bresenham<sup>1</sup> de rayon  $r$  autour d'un point candidat. Un coin est choisi s'il existe un nombre  $n$  de pixels contigus sur le cercle qui sont tous plus brillants (ou tous plus sombres) que le pixel candidat. Au même moment, Bay *et al* [13] ont proposé le détecteur « Fasthessian » qui se base sur une approximation de la matrice Hessienne.

---

<sup>1</sup> C'est un algorithme qui permet, pour une complexité algorithmique très réduite, de tracer des cercles en image numérique.

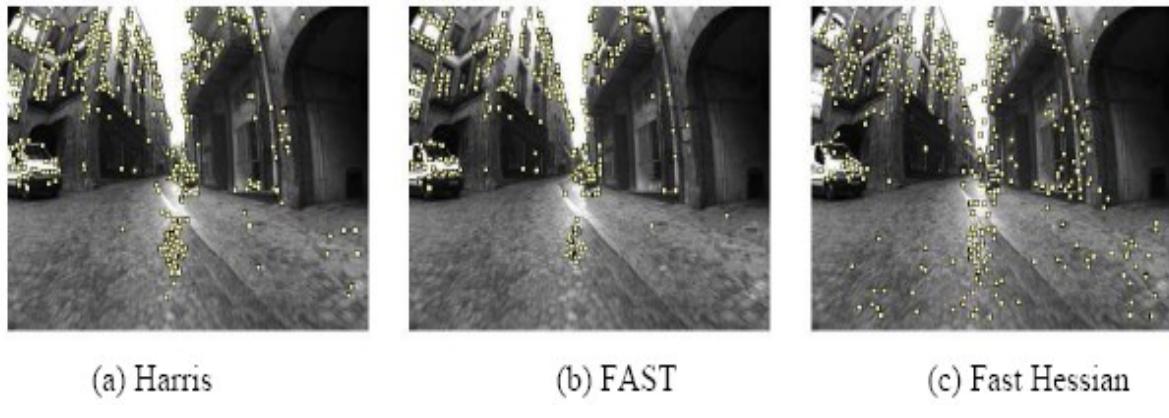


Figure 10 : Points d'intérêt détectés avec trois détecteurs différents

La figure 10 illustre un exemple d'image pour laquelle il a été appliqué trois détecteurs différents : Harris, Fast et Fast Hessian. Elle montre que le nombre de points d'intérêt détecté varie et que le détecteur FastHessian en détecte un plus grand nombre.

Le tableau 1 [24] représente une comparaison des détecteurs présentés. Il montre les différentes invariances qu'ils assurent ainsi que leurs performances face aux propriétés que doit satisfaire un point d'intérêt.

Tableau 1 : Comparaison de différents détecteurs de points d'intérêt

Détecteurs	Rotation Invariant	Scale Invariant	Affine Invariant	Répétabilité	Précision	Robustesse	Efficacité
Harris	X			+++	+++	+++	++
Hessian	X			++	++	++	+
SUSAN	X			++	++	++	+++
Harris-Laplace	X	X		+++	+++	++	+
Hessian-Laplace	X	X		+++	+++	+++	+
DoG	X	X		++	++	++	++
FastHessian	X	X		++	++	++	+++
Harris-Affine	X	X	X	+++	+++	++	++
Hessian-Affine	X	X	X	+++	+++	+++	++
MSER	X	X	X	+++	+++	++	+++

### 2.1.3. Présentation des principaux descripteurs de points d'intérêt

Un descripteur représente l'information contenue dans le voisinage d'un point d'intérêt. Plusieurs techniques ont été développées pour décrire des régions locales dans une image. Le descripteur le plus simple est celui représenté par un vecteur dont les composantes sont les pixels de l'image. La « Cross-corrélation » peut ensuite être utilisée pour calculer les scores

de similarité entre deux descripteurs. Cependant, les dimensions trop grandes de ces descripteurs induisent une complexité calculatoire très élevée.

Certains descripteurs sont basés sur la distribution et utilisent des histogrammes pour représenter les différentes caractéristiques d'apparences ou de formes. Lowe [7] proposa le « Scale Invariant Feature Transform » (SIFT), qui combine un détecteur invariant aux changements d'échelles et un descripteur basé sur la distribution du gradient dans la région détectée. Le descripteur, de dimension 128, est formé par un histogramme 3D ( $x, y, \theta$ ) des localisations des gradients et des orientations quantifiées. Le « Gradient Location-Orientation Histogram » (GLOH) de dimension 128 proposé dans [14] a été conçu pour améliorer la robustesse. Il considère une région spatiale pour le calcul de l'histogramme (3 bins dans la direction radiale et 8 bins dans la direction angulaire). C'est un descripteur très peu utilisé car très coûteux en temps de calcul. Le descripteur « Shape Context » proposé dans [22] se base sur les localisations des bords au lieu des localisations des gradients. Il a une dimension de taille 36. Le descripteur PCA-SIFT introduit dans [15]. Il est représenté par un vecteur des gradients de l'image dans les directions  $x$  et  $y$  calculés dans la région de support ; sa dimension est aussi égale à 36. Burghouts et Geusebroek [21] propose CSIFT. Le descripteur CSIFT est une extension du descripteur SIFT qui inclut les informations chromatiques de l'image.

D'autres descripteurs sont basés sur des techniques de fréquences spatiales et utilisent les fréquences contenues dans l'image. Pour cela la transformée de Fourier est utilisée pour décomposer le contenu de l'image dans des fonctions de base. Cependant, dans cette représentation les relations spatiales entre les points n'est pas explicite et les fonctions de base sont infinies dès lors il est difficile de les adapter aux approches locales.

Il existe aussi des descripteurs basés sur des techniques différentielles. Ces techniques se basent sur le calcul des dérivées des images à un certain ordre afin d'approximer le voisinage des points. Parmi ces descripteurs, Steerable Filters et Differential invariant [16,17] qui utilisent des dérivées calculées par convolution avec des dérivés gaussiennes d'écart types 6,7 sur un patch de l'image de taille 41. On peut citer aussi le descripteur Speed Up Robust Features (SURF) qui utilise la distribution des réponses des ondelettes de Haar dans le voisinage du point d'intérêt. C'est un descripteur invariant aux rotations et aux changements d'échelles et il est de dimension 64. La figure 11 montre le résultat de SURF sur une image de référence nommée Graffiti. Elle montre des fenêtres autour de différents points dont la taille et l'orientation dépend de l'échelle et de la direction dominante à lesquelles le point d'intérêt a été détecté.

Il y a aussi des descripteurs qui utilisent des vecteurs moments invariants. Ces moments ont été introduit par Van Gool *et al.* [18] pour décrire la nature multi-spectrale d'une image de données. Ils sont adaptés pour les images couleurs puisque les invariances peuvent être calculées pour chaque canal de couleurs et entre les canaux.

Très récemment, une autre méthode dénommée DAISY a été publiée par *Tola et al.* [19]. Il s'agit d'un descripteur local pour des appariements denses, qui possèdent un domaine d'exploration circulaire. Néanmoins, il reste couteux en temps de calcul.



Figure 11 : Image de type Graffiti montrant la taille et l'orientation de fenêtres de descripteurs à différentes échelles

#### 2.1.4. Evaluation des descripteurs

L'évaluation des descripteurs se basent sur la technique proposée dans [15]. Elle consiste à calculer le nombre d'appariements corrects et incorrects obtenus dans une paire d'images et le calcul des courbes imprécision-rappel.

Les courbes d'imprécision-rappel sont calculées en faisant varier un seuil, qui représente la distance maximale à ne pas dépasser entre deux descripteurs pour apparier deux régions situées dans deux images représentant la même scène.

Le taux de rappel est le nombre d'appariements corrects par le nombre de régions qui correspondent entre deux images représentant la même scène.

$$\text{rappel} = \frac{\text{nbre d'appariements corrects}}{\text{nbre de correspondances}} \quad (1)$$

Le taux d'imprécision est le nombre d'appariement incorrects par le nombre total d'appariements.

$$\text{imprécision} = \frac{\text{nbre d'appariements incorrects}}{\text{nbre d'appariements corrects} + \text{nbre d'appariements incorrects}}$$

Un descripteur parfait donnerait un rappel égal à 1 pour toutes précisions. En pratique, le rappel augmente si le seuil augmente ce qui engendre une augmentation du bruit due aux transformations qu'a subi l'image.

Dans les courbes d'imprécision-rappel, les traits horizontaux indiquent que le rappel est atteint avec une forte précision et est limité par les spécifications de la scène. Ceci veut dire que les structures détectées sont très similaires et que les descripteurs ne peuvent pas les distinguer. S'il y a une petite augmentation de la courbure ceci montre que le descripteur est affecté par la dégradation de l'image.

Les figures 13a et 13b extraite de [15] montrent respectivement les courbes d'imprécision-rappel de plusieurs types de descripteurs appliqués aux images 12a et 12b en conservant l'ordre de performance.

Les résultats montrent que tous les descripteurs sont affectés par la dégradation de l'image en appliquant un flou gaussien. Pour la première scène, GLOH, PCA-SIFT et SIFT ont le meilleur score. Dans la seconde scène, les résultats sont plus influencés par le flou appliqué. Les descripteurs ne peuvent pas distinguer les régions détectées vu que le flou appliqué les a rendus quasiment identique. Dans cette scène, SIFT donne le meilleur résultat alors que Cross-correlation donne le moins bon résultat malgré ces grandes dimensions. De



Figure 12 : Dégradation des images par flou gaussien

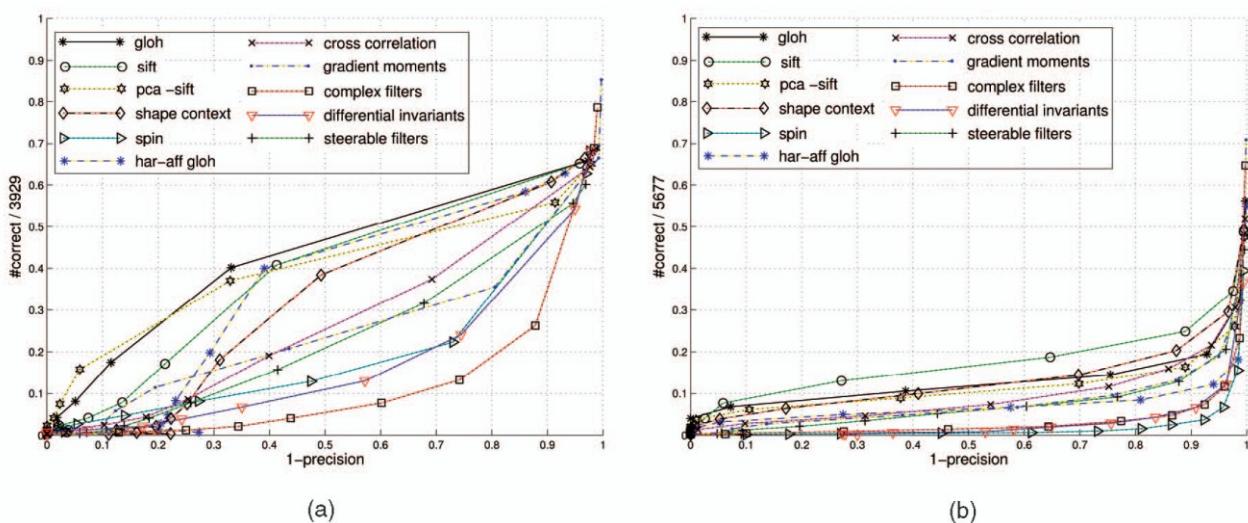


Figure 13 : Évaluation de 11 descripteurs par application de flou gaussien [15]

L'évaluation effectuée dans [15] ne compare pas SIFT à SURF. La figure 14 montre le résultat de trois descripteurs SIFT, CSIFT et SURF appliqués à des scènes d'images dégradées par variation du flou gaussien  $\beta$  [20].

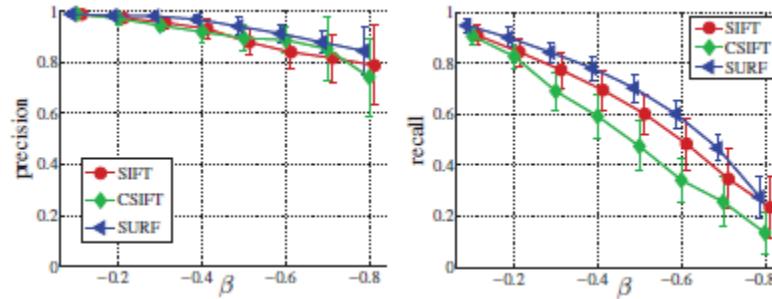


Figure 14 : Évaluation des descripteurs par application de flou pour trois types de descripteurs SIFT, CSIFT et SURF

Pour la précision les trois descripteurs sont à peu près équivalents. Cependant, pour le rappel SURF obtient le meilleur score suivi de SIFT et de CSIFT.

Le tableau 2 extrait de [20] montre le temps CPU moyen pour la détection et la description de points d'intérêt des algorithmes SIFT, CSIFT et SURF pour un Dual core Intel de fréquence 1,83 GHz. On peut constater aussi que la phase de détection est majoritaire en temps par rapport à la phase de description. D'autre part, le temps de calcul de SURF est de 24,31% inférieur à celui de SIFT. En effet, 0.1027 secondes sont nécessaires pour SURF alors que 0.1357 sont nécessaires pour SIFT.

Tableau 2 : Temps CPU moyen de SIFT, CSIFT et SURF pour la détection et la description des points d'intérêt

	Détection (Temps CPU)	Description (Temps CPU)	Temps CPU Total
SIFT	0.1276 secondes	0.0081 secondes	0.1357 secondes
CSIFT	0.3021 secondes	0.0757 secondes	0.3778 secondes
SURF	0.0854 secondes	0.0173 secondes	0.1027 secondes

## 2.2. Conclusion

L'analyse des images par la technique des descripteurs locaux nécessite deux phases. Une phase pour la détection des points d'intérêt et une seconde phase pour la description des points d'intérêt. L'état de l'art a montré qu'il existe beaucoup d'algorithmes respectant cette chaîne de traitement et que les algorithmes SIFT (Scale Invariant Feature Transform) et SURF (Speed Up Robust Features) sont les plus utilisés. SIFT et SURF sont les plus performants de point de vue performance et complexité. En effet, les points d'intérêts détectés sont robustes

vu qu'ils sont invariants aux déformations et aux bruits, de plus leurs descripteurs permettent d'apparier plusieurs points entre des images différentes représentant la même scène ou le même objet. D'autre part, la figure 14 et le tableau 2 montrent que SURF est plus performant que SIFT de point de vue performance et temps d'exécution.

### **3. Analyse fonctionnelle et calculatoire des algorithmes SIFT et SURF**

---

Ce chapitre représente une étude détaillée des algorithmes SIFT et SURF. Il est divisé en deux grandes parties. Une première partie dans laquelle est effectuée une analyse fonctionnelle et calculatoire de SIFT. Dans la seconde partie celle de SURF. L'analyse fonctionnelle des deux algorithmes permet de décrire précisément toutes les étapes de traitements dans la chaîne d'analyse d'image nécessaire pour la détection et la description des points d'intérêts. L'analyse calculatoire permet de déterminer le nombre et les types d'opérations effectuées pour aboutir aux détecteurs et aux descripteurs.

#### **3.1. Présentation de l'algorithme SIFT**

L'algorithme SIFT « Scale Invariant Feature Transform » a été proposé par David Lowe en 1999 pour détecter et décrire des zones d'intérêts dans une image [7]. C'est un algorithme invariant aux changements d'échelles puisque la détection des points se fait à différentes échelles. De plus, il est invariant aux rotations puisque pour chaque point d'intérêt détecté est affecté une orientation.

L'algorithme pour récupérer l'ensemble des traits caractéristiques se divise en deux grandes étapes :

- Détection :
  - o Détection des extrema dans l'espace échelle
  - o Localisation des points d'intérêt
- Description :
  - o Affectation de l'orientation
  - o Calcul du descripteur

##### **3.1.1. Détection des extrema dans l'espace échelle**

Le principe de la détection consiste à trouver les extrema dans l'espace-échelle gaussien qui est défini comme une convolution d'une variable d'échelle gaussienne.

L'espace échelle gaussien est divisé en un nombre d'octaves. Une octave représente un ensemble de niveaux consécutifs. Ces niveaux consécutifs sont le résultat de filtrage de l'image de départ par une cascade de filtre gaussien séparés par un facteur  $k$  (voir figure 17). L'image de départ est considérée comme ayant une échelle  $\sigma$ . A chaque changement d'octave,

l'échelle double de taille. Pour les niveaux d'une octave, les échelles sont étalées entre  $\sigma$  et  $2\sigma$  c'est-à-dire suivant une suite géométrique de raison :

$$k = 2^{\frac{1}{\text{nbre de niveaux par octave}}} \quad (3)$$

Cette espace échelle gaussien est défini par  $L(x, y, \sigma)$ , la variable gaussienne  $G(x, y, \sigma)$  et l'image  $I(x, y)$ .

Avec :

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (4)$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (5)$$

Dans le but de détecter des points d'intérêt stables, *Lowe* [7] propose d'utiliser la différence de gaussienne DoG. Cette DoG est calculée sur deux échelles différentes séparées par le facteur k.

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma) \quad (6)$$

La différence de gaussienne fournit une approximation du Laplacien de Gauss normalisé  $\sigma^2 \nabla^2 G$ . Les maximums de cette fonction produisent les points saillants, les plus stables :

$$\sigma^2 \nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \quad (7)$$

La figure 15 illustre le cas d'un espace échelle gaussien constitué de 4 octaves et trois niveaux.

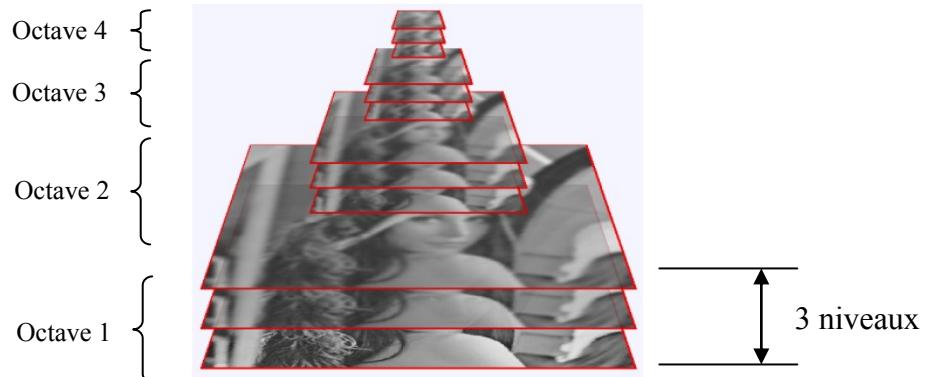


Figure 15 : Pyramide d'une image représentée dans un espace-échelle gaussien de 4 octaves et 3 niveaux

Les extrema locaux des DoG sont détectés par comparaison avec ses 26 voisins. Les 8 voisins à la même échelle et les 9+9 aux niveaux précédents et suivants. Ainsi un point est détecté s'il est plus grand que ses 26 voisins comme le montre la figure 16.

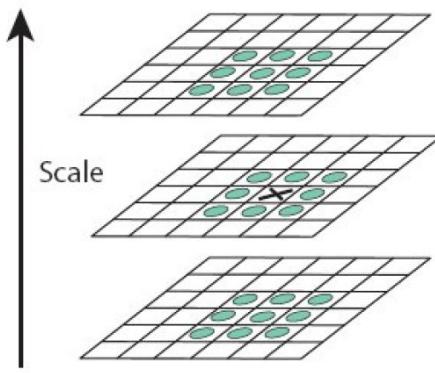


Figure 16 : Recherche des extremums

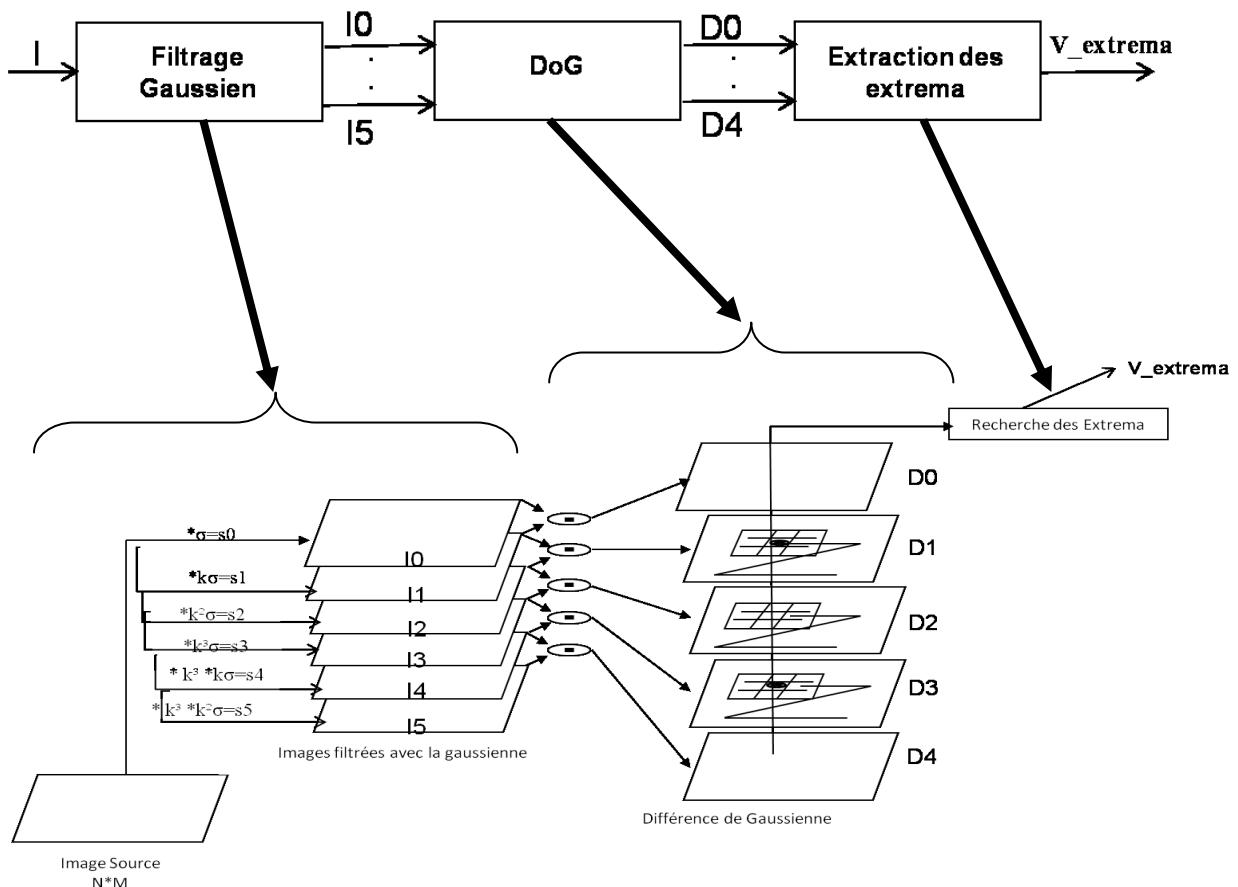


Figure 17 : Détection des extrema dans un espace échelle gaussien constitué d'une octave à 6 niveaux

### 3.1.2. Localisation des points d'intérêt

Une fois les extrema détectés, un certain nombre d'heuristiques sont calculées pour supprimer les points correspondant à du bruit ou localisés le long des lignes (figure 18). Pour cela, on fait correspondre aux points locaux une fonction quadratique 3D pour interpoler la position des maxima locaux.

Ainsi en faisant une expansion de Taylor :

$$D(X) = D + \frac{\partial D^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial^2 X} X \quad (8)$$

Avec  $X = (x, y, \sigma)^T$

$D(X)$  le point de coordonnée  $(x, y)$  dans la DoG d'échelle  $\sigma$

Ainsi l'extremum  $\hat{X}$  est calculé en prenant la dérivée et en faisant passer  $X$  par 0 :

$$\hat{X} = - \frac{\partial^2 D^{-1}}{\partial X^2} \frac{\partial D}{\partial X} \quad (9)$$

Avec :

$$\frac{\partial^2 D}{\partial X^2} = \begin{pmatrix} \frac{\partial^2 D(x, y, \sigma)}{\partial x^2} & \frac{\partial^2 D(x, y, \sigma)}{\partial x \partial y} & \frac{\partial^2 D(x, y, \sigma)}{\partial x \partial \sigma} \\ \frac{\partial^2 D(x, y, \sigma)}{\partial x \partial y} & \frac{\partial^2 D(x, y, \sigma)}{\partial y^2} & \frac{\partial^2 D(x, y, \sigma)}{\partial y \partial \sigma} \\ \frac{\partial^2 D(x, y, \sigma)}{\partial x \partial \sigma} & \frac{\partial^2 D(x, y, \sigma)}{\partial y \partial \sigma} & \frac{\partial^2 D(x, y, \sigma)}{\partial \sigma^2} \end{pmatrix} \quad (10)$$

$$\frac{\partial D}{\partial X} = \begin{pmatrix} \frac{\partial D(x, y, \sigma)}{\partial x} \\ \frac{\partial D(x, y, \sigma)}{\partial y} \\ \frac{\partial D(x, y, \sigma)}{\partial \sigma} \end{pmatrix} \quad (11)$$

Voir Annexe B pour le détail de calcul de chaque composante de (10) et (11).

La fonction  $D(\hat{X})$  est utilisée pour rejeter les points instables.

Avec  $D(\hat{X})$  de la forme :

$$D(\hat{X}) = D + \frac{1}{2} \frac{\partial D^T}{\partial X} \hat{X} \quad (12)$$

Les points rejetés sont les points pour lesquels  $|D(\widehat{X})| < 0,03$  [12].



Figure 18 : Localisation des points d'intérêt

### 3.1.3. Affectation de l'orientation

Une orientation dominante est assignée à chaque point d'intérêt détecté en se basant sur les propriétés locales de l'image.

La figure 19 illustre les différentes étapes nécessaires pour déterminer la direction dominante d'un point d'intérêt. La première étape consiste à calculer la norme (équation 13) et l'angle (équation 14) des gradients se trouvant dans le voisinage de taille 16x16 autour du point d'intérêt détecté.

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (13)$$

$$\theta(x, y) = \tan^{-1} \left( \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \right) \quad (14)$$

Avec  $L(\sigma)$  l'image à l'échelle  $\sigma$  et  $L(x, y, \sigma)$  le point de coordonnée  $(x, y)$  dans l'image d'échelle  $\sigma$ .

Une fois la norme et l'angle calculés, la seconde étape consiste à construire l'histogramme des orientations des gradients. Cet histogramme est constitué de 36 régions couvrant une rangée d'orientation de 360 degrés, par pas de 10° ( $\theta$  prend les valeurs  $[0^\circ, 10^\circ, 20^\circ, \dots, 350^\circ]$ ). De plus, chaque point du voisinage est pondéré par une fenêtre gaussienne de taille  $1,5\sigma$ . L'équation (15) montre la valeur qu'il faut ajouter à chaque région de l'histogramme.

$$\text{hist}(\theta) = \sum_{pixels \in secteur[\theta, \theta+10]} m * \text{Gauss}(\text{dist(pixel, pt d'intérêt)}, 1,5\sigma) \quad (15) \quad (\text{Sans recouvrement}).$$

Les pics présents dans l'histogramme correspondent aux différentes orientations. Le pic le plus élevé correspond à la direction dominante. Pour les pics ayant une valeur égale à 80% de la valeur du pic dominant, un autre point d'intérêt est considéré avec cette orientation.

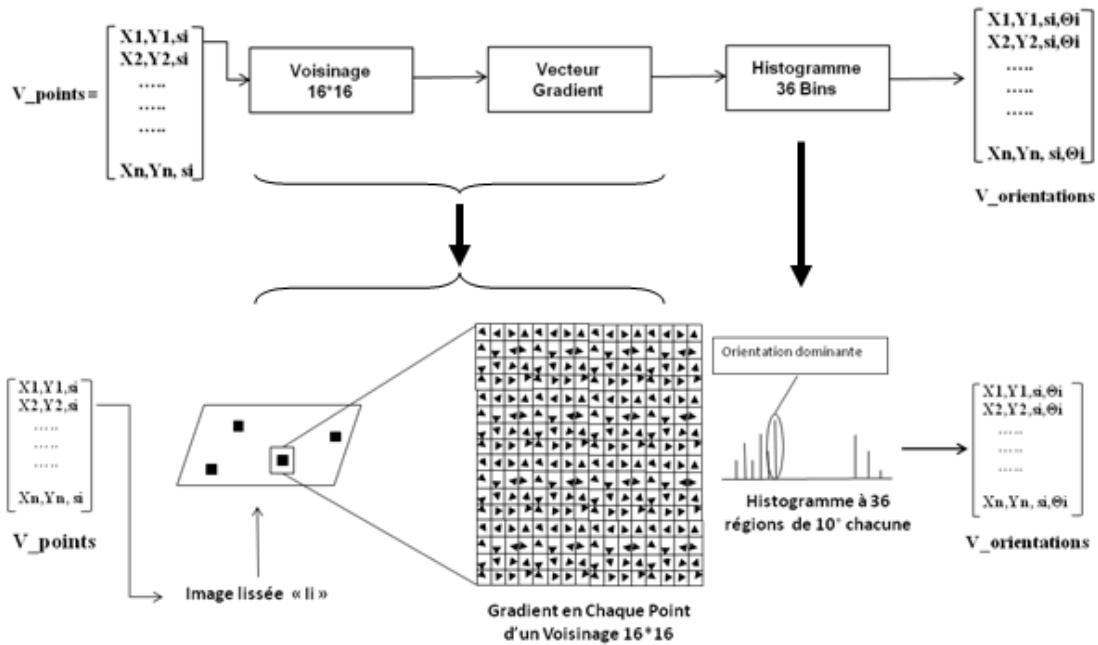


Figure 19 : Étapes de calcul de l'orientation du point d'intérêt

### 3.1.4. Calcul du descripteur

Plusieurs étapes sont nécessaires pour le calcul du descripteur. Elles sont illustrées par la figure 20. La première consiste à considérer une région de taille 16x16 autour du point d'intérêt orienté suivant la direction dominante. Ensuite, cette région est découpée en 16 sous-régions de taille 4x4 chacune. Pour chaque sous-région la norme et l'orientation du gradient sont calculées. Ces normes et ces orientations sont utilisées pour construire un histogramme à 8 régions couvrant une rangée d'orientation de 360 degrés ainsi  $\theta$  prend les valeurs  $[0^\circ, 45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 270^\circ, 315^\circ]$

De plus, une pondération est assignée à la norme du gradient pour les régions voisines au sein d'un même histogramme [12]. L'équation 16 montre la valeur de cette pondération.

$$v = m * W_{win}(x, y) * W_{carre}(x, y) \quad (16)$$

$$W_{win}(x, y) = \text{Gauss}((x, y), 8) \quad (17)$$

$$W_{carre}(x, y) = \max(1 - dx) * \max(1 - dy) \quad (18)$$

Avec  $dx$  et  $dy$  les distances normalisées au centre de la sous-région considérée.

Ainsi pour  $\theta$  compris entre la Région 0 et la Région 1 :

$$\text{hist}(\text{Région } 0) = \text{hist}(\text{Région } 0) + v * \frac{\text{Région } 1 - \theta}{\text{Région } 1 - \text{Région } 0} \quad (19)$$

$$\text{hist}(\text{Région } 1) = \text{hist}(\text{Région } 1) + v * \frac{\theta - \text{Région } 0}{\text{Région } 1 - \text{Région } 0} \quad (20)$$

Une fois l'histogramme de toutes les sous-régions calculées, le descripteur SIFT est construit. Ce dernier contient les valeurs extraites des histogrammes de chaque sous-région. Ainsi il a une dimension 128 puisque la région qui entoure le point d'intérêt est découpée en 16 sous-régions contenant chacune 8 orientations différentes.

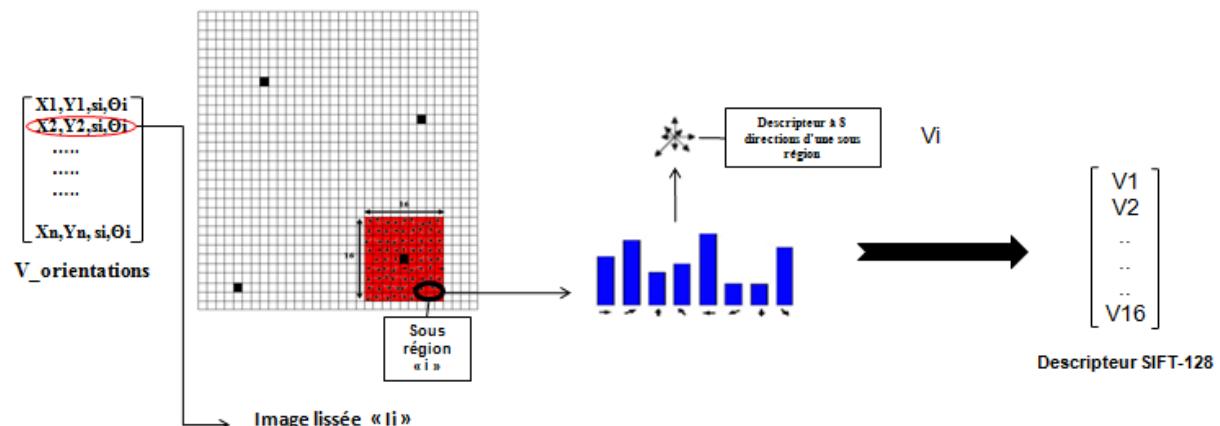
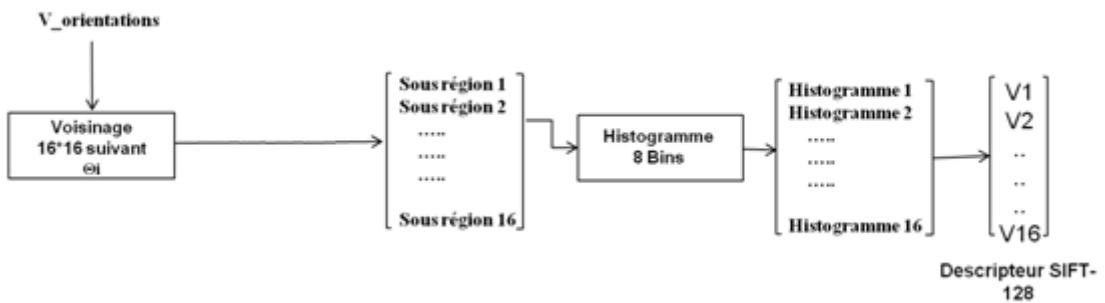


Figure 20 : Étapes de construction du descripteur SIFT

### 3.2. Analyse calculatoire de SIFT

Dans cette partie, nous allons détailler la complexité calculatoire des quatre grandes étapes de l'algorithme SIFT pour une image de taille NxM pixels. Pour cela, nous allons calculer le nombre d'opérations utilisées dans les équations présentées dans l'analyse fonctionnelle de SIFT.

- **Complexité calculatoire de l'étape détection des extrema dans l'espace échelle**

Une cascade de filtres gaussiens de taille  $2^{octave+1}k^{niveau}\sigma$  est appliquée en chaque point de l'image. Ainsi, le nombre d'opérations nécessaire pour l'étape de lissage d'une image est :

$$Nbre\ d'additions = (2^{octave+1} * k^{niveau} * \sigma - 1) * 2^{octave+1} * k^{niveau} * \sigma * N * M$$

$$Nbre\ de\ multiplications = ((2^{octave+1} * k^{niveau} * \sigma) * (2^{octave+1} * k^{niveau} * \sigma)) * N * M$$

$$Nbre\ de\ lectures = ((2^{octave+1} * k^{niveau} * \sigma * 2) * (2^{octave+1} * k^{niveau} * \sigma)) * N * M$$

La prochaine étape est le calcul des DoG qui nécessite :

- $Nbre\ d'additions = (N * M)$
- $Nbre\ de\ lectures = 2 * N * M$

Enfin, la dernière étape consiste à extraire les extrema qui nécessitent :

- $26\ comparaisons * (N - 1) * (M - 1)$

- **Complexité calculatoire de l'étape localisation des points d'intérêt**

Une étude des équations présentées dans l'annexe B a permis de déterminer le nombre d'opérations nécessaires dans l'équation (10). Ce nombre d'opération est répété pour chaque extrema détecté :

- $Nbre\ d'additions = 18$
- $Nbre\ de\ décalages = 18$
- $Nbre\ de\ lectures = 27$

L'équation (9) a donc besoin :

- $Nbre\ d'additions = 54$
- $Nbre\ de\ décalages = 18$
- $Nbre\ de\ multiplications = 72$
- $Nbre\ de\ lectures = 135$
- $Nbre\ de\ divisions = 1$

- **Complexité calculatoire de l'étape affectation de l'orientation**

Le calcul de l'orientation de chaque point d'intérêt s'effectue en utilisant les équations (13) et (14) pour calculer la norme du gradient.

L'équation (13) nécessite :

- *Nbre d'additions = 3*
- *Nbre de multiplications = 2*
- *Nbre d'accès mémoires = 4*
- *complexité de la racine*

L'équation (14) nécessite :

- *Nbre d'additions = 2*
- *Nbre de divisions = 1*
- *Nbre d'accès mémoires = 4*
- *complexité de l'arctangente*

Les équations (13) et (14) sont appliquées à 256 points vu qu'un voisinage de taille 16x16 est considéré pour chaque point d'intérêt.

Ensuite, pour la construction de l'histogramme une pondération par une fenêtre gaussienne de la norme et de l'angle est effectuée. Cette pondération est calculée en utilisant l'équation 15.

Cette équation nécessite :

- *Nbre d'additions =  $(3\sigma - 1) * 3\sigma + 1$*
- *Nbre de multiplications =  $(3\sigma * 3\sigma) + 1$*
- *Nbre de lectures =  $(3\sigma * 2) * 3\sigma + 2$*

#### • Complexité calculatoire de l'étape Calcul du descripteur

Le calcul du descripteur s'effectue par l'intermédiaire d'un histogramme. Cet histogramme est construit pour chaque point d'intérêt en utilisant :

- L'équation (16) pour calculer la pondération assignée à la norme du gradient pour les régions voisines de l'histogramme.
- Les équations (19) et (20) sont utilisées pour le calcul des composantes de l'histogramme.

L'équation (16) nécessite :

- *Nbre d'additions =  $(16\sigma - 1) * 16\sigma + 2$*
- *Nbre de multiplications =  $(16\sigma * 16\sigma) + 3$*
- *Nbre de lectures =  $(16\sigma * 2) * 16\sigma + 3$*
- *Nbre de comparaisons = 2*

Les équations (19) et (20) nécessitent chacune :

- *Nbre d'additions = 3*
- *Nbre de multiplications = 1*
- *Nbre d'accès mémoires = 5*

- *Nbre de divisions = 1*

Les équations (16), (19) et (20) vont être appliquées à 256 points vu qu'un voisinage de taille 16x16 est considéré pour chaque point d'intérêt.

### **3.3. Bilan de l'analyse de l'algorithme SIFT**

La phase de détection des points d'intérêt est très coûteuse en ce qui concerne le nombre d'opérations par rapport à la phase de description. Ceci est largement dû aux cascades de filtres gaussiens appliqués à l'image de départ pour déterminer les images lissées et assurer ainsi l'invariance aux changements d'échelles et ceux utilisés pour déterminer l'orientation dominante de chaque point d'intérêt afin d'assurer l'invariance de rotation.

### **3.4. SURF**

L'algorithme SURF a été proposé par Herbert Bay en 2006 pour détecter et décrire des points d'intérêt dans une image [13]. Il utilise le détecteur FastHessien pour l'extraction des points d'intérêt et un descripteur qui repose sur une distribution des ondelettes de Haar dans le voisinage des points d'intérêt. C'est un algorithme invariant aux changements d'échelles puisque la détection des points se fait à différentes échelles. En plus, il est invariant aux rotations puisque pour chaque point d'intérêt détecté est affecté une orientation.

L'approche adoptée dans l'algorithme SURF pour récupérer l'ensemble des descripteurs peut être divisée en trois grandes étapes (voir figure 21) :

- Calcul de l'image Intégrale
- Extraction des points d'intérêts avec le détecteur « Fast-Hessian »
  - o La matrice Hessienne
  - o Construction de l'espace-échelle
  - o Localisation des points d'intérêts
- Calcul du descripteur
  - o Calcul de l'orientation dominante
  - o Calcul des composantes du descripteur

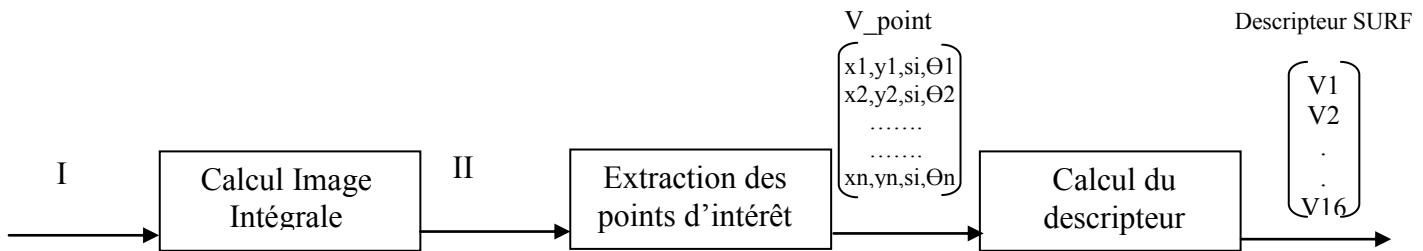


Figure 21 : Étapes de l'algorithme SURF

### 3.4.1. L'image intégrale

L'image intégrale notée  $ii(x, y)$  à la localisation  $(x, y)$  contient la somme des valeurs des pixels situés au-dessus et à gauche du point  $(x, y)$  [23] :

$$ii(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y'), \quad (21)$$

Où  $i(x, y)$  est l'image d'entrée.

Le calcul de l'image intégrale peut s'effectuer en une seule passe en utilisant les formules de récurrence suivantes :

$$s(x, y) = s(x, y - 1) + i(x, y) \quad (22)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \quad (23)$$

Avec  $s(x, y)$  l'accumulation des valeurs des pixels dans la ligne et  $s(x, -1) = ii(-1, y) = 0$ .

En utilisant l'image intégrale, trois additions et quatre accès mémoires sont nécessaires pour calculer la somme des intensités à l'intérieur d'une région rectangulaire comme est illustré par la figure 22.

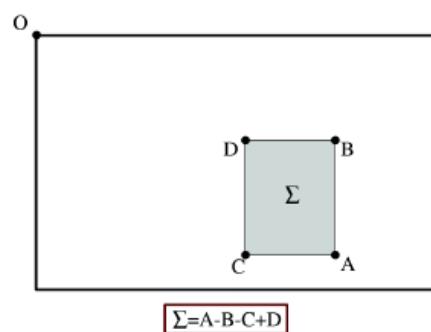


Figure 22 : Calcul de surface en utilisant l'image intégrale

L'inconvénient majeur de l'image intégrale est sa dynamique qui dépend du contenu de l'image. En effet, l'image intégrale est une accumulation. Ainsi dans le pire cas, pour une image de taille 640x480 constituées de pixels ayant chacun une valeur égale à 255, 27 bits sont nécessaire pour stocker un pixel de l'image intégrale.

### 3.4.2. Le détecteur FastHessian

Le détecteur de SURF est basé sur le calcul du déterminant de la matrice Hésienne. Pour un point  $X = (x, y)$  dans une image  $I$ , la matrice Hésienne,  $H(X, \sigma)$ , au point  $X$  à l'échelle  $\sigma$  est définie par :

$$H(X, \sigma) = \begin{pmatrix} L_{xx}(X, \sigma) & L_{xy}(X, \sigma) \\ L_{xy}(X, \sigma) & L_{yy}(X, \sigma) \end{pmatrix} \quad (24)$$

Avec :

$$L_{xx}(X, \sigma) = \frac{\partial^2 G(x, y, \sigma)}{\partial x^2} * I(x, y) \quad (25)$$

$$L_{xy}(X, \sigma) = \frac{\partial^2 G(x, y, \sigma)}{\partial x \partial y} * I(x, y) \quad (26)$$

$$L_{yy}(X, \sigma) = \frac{\partial^2 G(x, y, \sigma)}{\partial y^2} * I(x, y) \quad (27)$$

Et

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Le déterminant de la matrice Hésienne est défini par :

$$\det(H(X, \sigma)) = L_{xx}L_{yy} - L_{xy}^2 \quad (28)$$

Herbert Bay [13], propose d'approximer le Laplacien par des « boites filtres ». La figure 23 illustre la similarité entre ces boites filtres et les dérivées secondees du gaussien discrétilisées et recadrées. Cette approximation a apporté une amélioration considérable de la performance surtout lorsqu'elle est calculée avec l'image intégrale. En effet, pour un filtre de taille 9x9, les filtres réels nécessitent 81 accès mémoires et opérations. Cependant, « les boites filtres » nécessitent seulement 8 accès mémoires et opérations. De plus, le coût de calcul n'augmente pas si la taille de ces boites filtres augmente, ce qui n'est pas le cas des filtres originaux représentants le Laplacien.

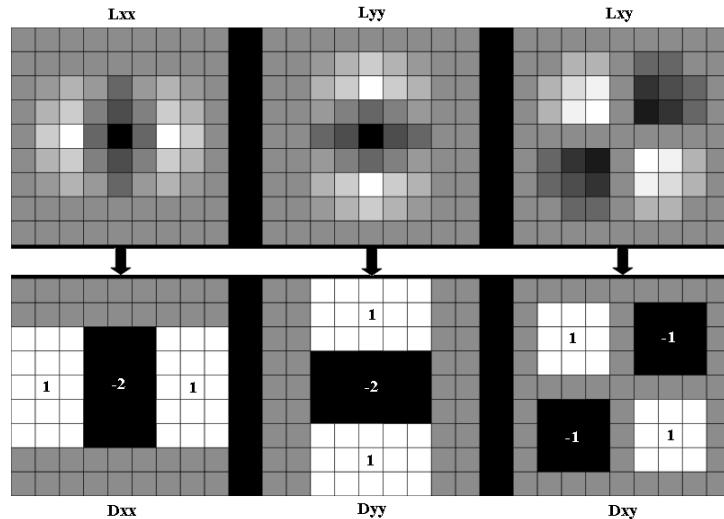


Figure 23 : Approximation du Laplace des gaussiennes

La figure 23 montre l'approximation de  $L_{xx}$ ,  $L_{yy}$  et  $L_{xy}$  par  $D_{xx}$ ,  $D_{yy}$  et  $D_{xy}$ .

Herbert Bay [13], propose l'équation (29) comme une approximation précise du déterminant de la matrice Hésienne :

$$\det(H_{approx}) = D_{xx}D_{yy} - (0,9D_{xy})^2 \quad (29)$$

### 3.4.3. Construction de l'espace-échelle

L'espace-échelle dans SURF est créé en augmentant la taille des noyaux appliqués à l'image. Ceci permet de calculer de manière simultanée plusieurs niveaux de la pyramide d'où l'amélioration de la performance. L'espace-échelle est divisé en un nombre d'octaves et l'octave regroupe un ensemble de niveaux. Le niveau le plus bas est le résultat de convolution des filtres de taille 9x9 avec l'image, tel que ces filtres correspondent à une gaussienne de valeur  $\sigma=1.2$ .

L'équation ci-dessous représente la formule utilisée pour calculer l'échelle  $\sigma_{approx}$ .

$$\sigma_{approx} = \text{taille du filtre courant} * \frac{\text{échelle du filtre de base}}{\text{taille du filtre de base}} \quad (30)$$

Le tableau 3 montre les valeurs des échelles ainsi que la taille des filtres associés à chaque niveau. Les tailles des filtres ont été proposées par Herbert Bay [13].

Tableau 3 : Valeurs des échelles à travers les octaves et les niveaux

Octave	1				2			
Niveau	1	2	3	4	1	2	3	4
Taille du Filtre	9	15	21	27	15	27	29	51
$\sigma_{approx}$	1.2	2	2.8	3.6	2	3.6	5.2	6.8

La figure 24 illustre les différentes étapes qui mènent à extraire les points candidats dans le cas d'une octave à quatre niveaux. Ces points candidats sont situés dans les matrices 1, 2, 3 et 4 et représentent le résultat (21). Les composantes de l'équation (21) sont le résultat de convolution des « boîtes filtres » avec l'image intégrale c'est-à-dire les réponses des filtres 9x9, 15x15, 21x21 et 27x27.

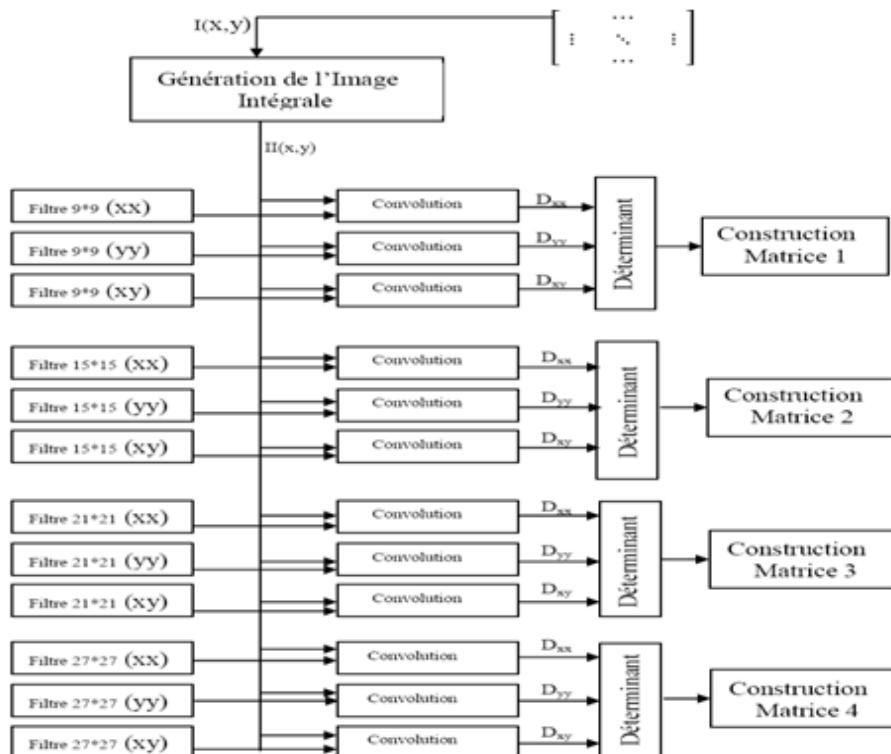


Figure 24 : Les étapes d'extraction des points candidats

### 3.4.4. Localisation des points d'intérêts

La localisation des points d'intérêts peut être divisée en trois phases. La première phase consiste à éliminer tous les points inférieurs à un seuil. Cette phase est optionnelle, elle est utilisée dans le but de diminuer le nombre de points à décrire donc diminuer le nombre d'opérations dans la phase calcul du descripteur. La seconde phase, consiste à effectuer une suppression des non maxima et des non minima comme l'illustre la figure 16.

La dernière phase consiste à déterminer l'échelle et les coordonnées exactes des extrema détectés, c'est-à-dire sa localisation dans l'image lissée d'où il provient (résultat de convolution entre l'image intégrale et les boîtes filtres). Pour cela, SURF utilise également l'expansion de Taylor :

$$I(X) = I + \frac{\partial I}{\partial X}X + \frac{1}{2}X^T \frac{\partial^2 I}{\partial^2 X}X \quad (31)$$

Où  $X = (x, y, \sigma)^T$

$I(X)$  est le point de coordonnée  $(x, y)$  dans l'image  $I$  d'échelle  $\sigma$ .

En prenant la dérivée de cette fonction, la localisation  $\hat{X}$  de l'extrema est obtenue tel que :

$$\hat{X} = -\frac{\partial^2 I^{-1}}{\partial X^2} \frac{\partial I}{\partial X} \quad (32)$$

Avec :

$$\frac{\partial^2 I}{\partial X^2} = \begin{pmatrix} \frac{\partial^2 I(x, y, \sigma)}{\partial x^2} & \frac{\partial^2 I(x, y, \sigma)}{\partial x \partial y} & \frac{\partial^2 I(x, y, \sigma)}{\partial x \partial \sigma} \\ \frac{\partial^2 I(x, y, \sigma)}{\partial x \partial y} & \frac{\partial^2 I(x, y, \sigma)}{\partial y^2} & \frac{\partial^2 I(x, y, \sigma)}{\partial y \partial \sigma} \\ \frac{\partial^2 I(x, y, \sigma)}{\partial x \partial \sigma} & \frac{\partial^2 I(x, y, \sigma)}{\partial y \partial \sigma} & \frac{\partial^2 I(x, y, \sigma)}{\partial \sigma^2} \end{pmatrix} \quad (33)$$

$$\frac{\partial I}{\partial X} = \begin{pmatrix} \frac{\partial I(x, y, \sigma)}{\partial x} \\ \frac{\partial I(x, y, \sigma)}{\partial y} \\ \frac{\partial I(x, y, \sigma)}{\partial \sigma} \end{pmatrix} \quad (34)$$

Voir Annexe B pour le détail de calcul de chaque composante de (33) et (34).

La fonction  $I(\hat{X})$  est utilisée pour rejeter les points instables. Avec  $I(\hat{X})$  de la forme :

$$I(\hat{X}) = D + \frac{1}{2} \frac{\partial I^T}{\partial X} \hat{X} \quad (35)$$

Les points rejetés sont les points pour lesquels  $|I(\hat{X})| < 0,5$  [13].

### 3.4.5. Calcul de l'orientation dominante

La figure 25 montre les différentes étapes qui permettent de déterminer l'orientation dominante d'un point d'intérêt. Tout d'abord, une région de rayon  $6\sigma$  est construite autour de chaque point d'intérêt. Pour chaque point  $(x_i, y_i)$  de la région, le produit de convolution entre la gaussienne de taille  $2\sigma$  et la réponse des ondelettes de Haar de taille  $4\sigma$  selon la direction x des abscisses et la direction y des ordonnées du point  $(x_i, y_i)$  est calculée. La figure 26 montre les ondelettes de Haar selon la direction de x et selon la direction de y.

Après le parcours de tous les points, deux vecteurs  $ResX$  et  $ResY$  sont obtenus, ces deux vecteurs représentent une pondération des réponses des ondelettes de Haar.

$$ResX(x_i, y_i) = Gauss((x_i, y_i), 2\sigma) * HaarX((x_i, y_i), 4\sigma) \quad (36)$$

$$ResY(x_i, y_i) = Gauss((x_i, y_i), 2\sigma) * HaarY((x_i, y_i), 4\sigma) \quad (37)$$

Une fois les réponses de Haar calculées, une fenêtre de  $\pi/3$  avec un pas de valeur  $\pi/32$  est glissée sur la région circulaire entourant le point d'intérêt en sommant à chaque fois les composantes des vecteurs  $ResX$  et  $ResY$  se trouvant dans la fenêtre. L'orientation dominante est la région où cette somme est maximale.

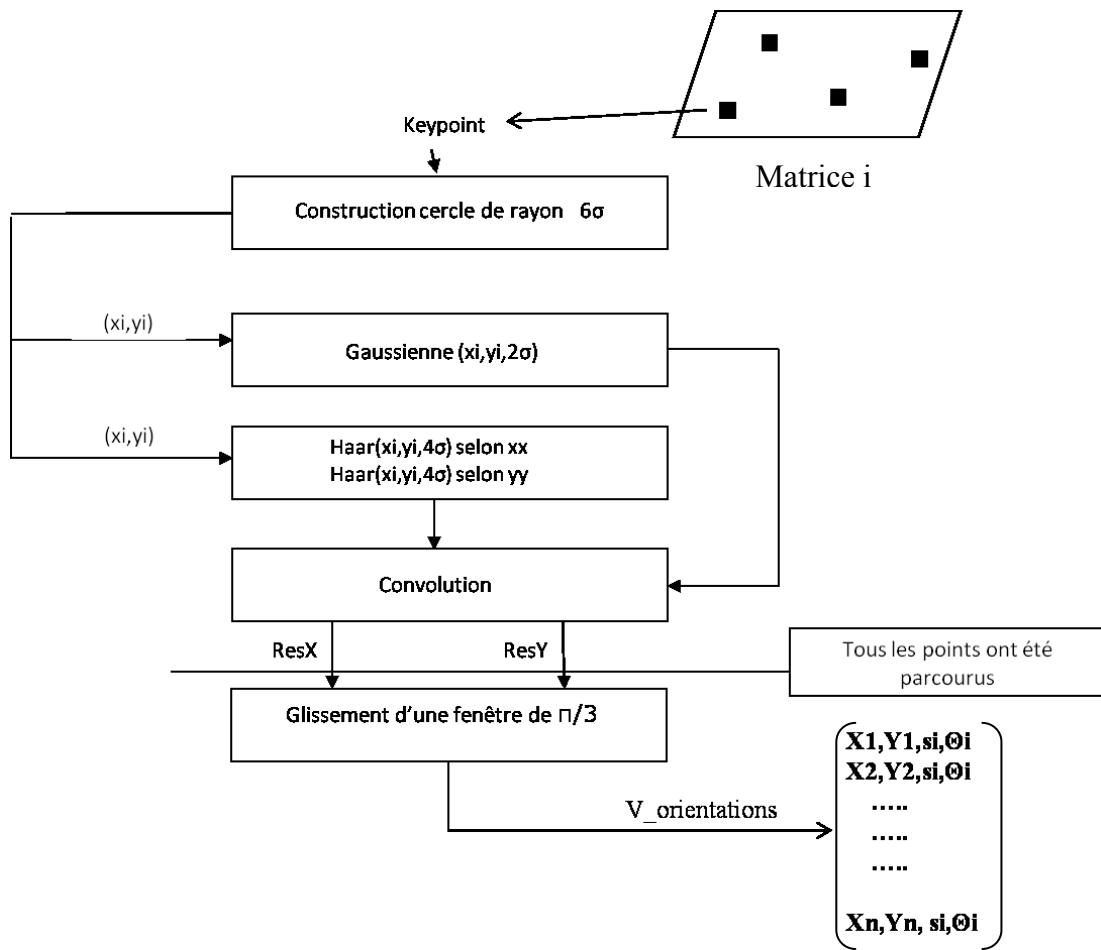


Figure 25 : Étapes nécessaires pour déterminer l'orientation

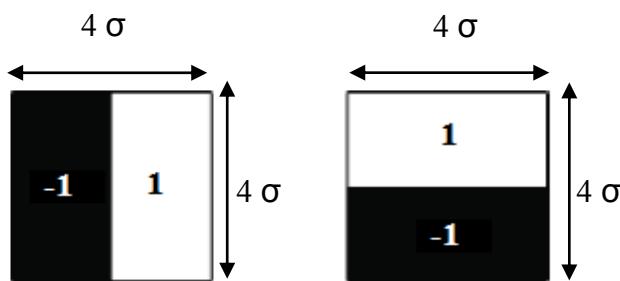


Figure 26 : A gauche filtre de Haar selon xx et à droite filtre de Haar selon yy

### 3.4.6. Calcul des composantes du(descripteur)

Pour calculer les composantes, une région de taille  $20\sigma$  est construite autour du point d'intérêt. Une fois cette région construite, elle est découpée en 16 sous régions de taille  $5\sigma$  chacune. Ensuite, pour chaque point d'une sous-région, le produit de convolution entre la gaussienne de taille  $3,3\sigma$  et la réponse des ondelettes de Haar de taille  $2\sigma$  selon la direction x et la direction y est calculé (équations (38) et (39)).

Une fois que les réponses de Haar de tous les points des 16 sous régions ont été calculées, un vecteur contenant la somme des valeurs des réponses de Haar selon x et selon y ainsi que la somme des valeurs absolues des réponses de Haar selon x et selon y est calculé (équation (40)). Ce vecteur constitue le descripteur de cette sous-région. Le descripteur SURF regroupe l'ensemble de ces vecteurs. Ainsi, la taille du descripteur SURF est égale à 64 mots de 32 bits (16 sous-régions x 4 paramètres). Ce descripteur caractérise la région qui entoure le point d'intérêt.

La figure 27 illustre ces différentes étapes menant à la construction du descripteur SURF-64.

$$Res_1X(x_i, y_i) = \text{Gauss}((x_i, y_i), 3,3\sigma) * \text{Haar}X((x_i, y_i), 2\sigma) \quad (38)$$

$$Res_1Y(x_i, y_i) = \text{Gauss}((x_i, y_i), 3,3\sigma) * \text{Haar}Y((x_i, y_i), 2\sigma) \quad (39)$$

$$V^T = (\sum Res_1X(x_i, y_i), \sum Res_1Y(x_i, y_i), \sum |Res_1X(x_i, y_i)|, \sum |Res_1Y(x_i, y_i)|) \quad (40)$$

Avec :

$$\sum Res_1X(x_i, y_i) = \sum dx$$

$$\sum Res_1Y(x_i, y_i) = \sum dy$$

$$\sum |Res_1X(x_i, y_i)| = \sum |dx|$$

$$\sum |Res_1Y(x_i, y_i)| = \sum |dy|$$

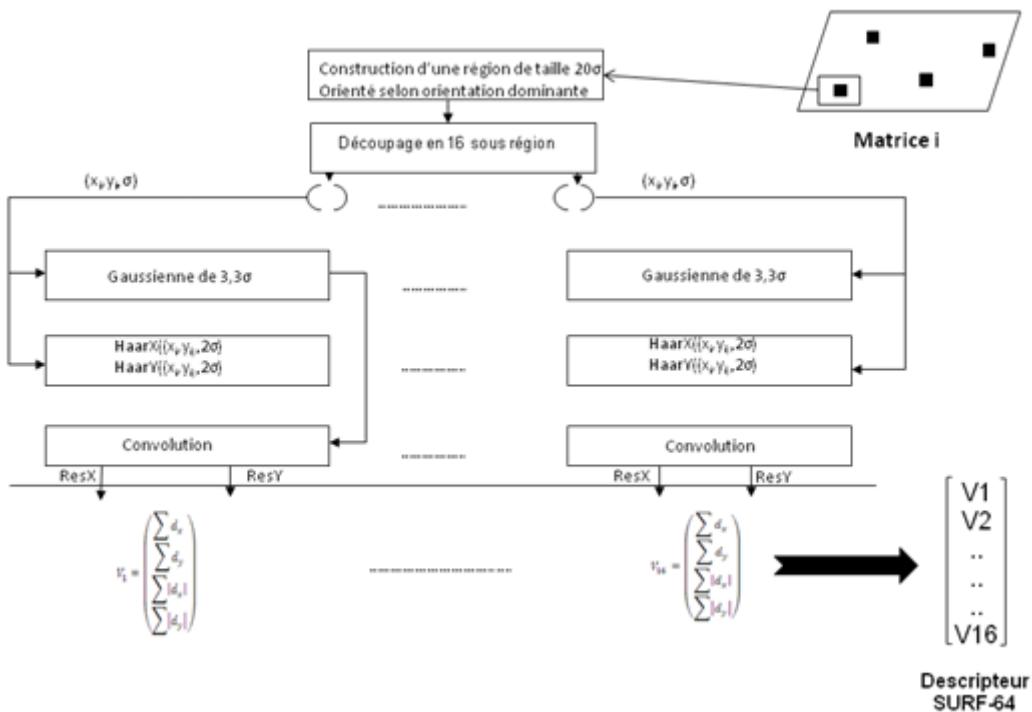


Figure 27 : Étapes nécessaires pour la construction du descripteur SURF

### 3.5. Analyse calculatoire de SURF

Dans cette partie, nous allons détailler la complexité calculatoire de l'algorithme SURF pour une image de taille  $N \times M$  pixels. Pour cela, nous allons calculer le nombre d'opérations utilisées dans les équations présentées dans l'analyse fonctionnelle de SURF.

- **Complexité calculatoire de l'étape calcul de l'image Intégrale**

Le calcul de l'image intégrale s'effectue avec les équations (22) et (23) qui ont besoin :

- $Nbre d'additions = 2 * N * M$
- $Nbre d'accès mémoires = N * M * p + N$

- **Complexité calculatoire du détecteur « FastHessian »**

Pour un algorithme SURF constitué d'une pyramide à 2 octaves et 4 échelles  $6 \times 3 = 18$  « boites filtres » sont nécessaires (voir tableau 3).

Le produit de convolution entre un « boite filtre » selon la direction x et selon la direction y et l'image intégrale nécessite :

- $Nbre d'additions = 8xNxM$
- $Nbre de multiplications = 4xNxM$
- $Nbre de lectures = 8xNxM$

Le produit de convolution entre un « boite filtre » selon la direction xy et l'image intégrale nécessite :

- $Nbre d'additions = 15xNxM$
- $Nbre de lectures = 15xNxM$

Donc, le nombre d'opérations nécessaire pour déterminer l'ensemble des points candidats (Figure 20) est :

- $Nbre d'additions = 6x(8xNxM)x2 + 6x(15xNxM) + 6xNxM$
- $Nbre de multiplications = 6x(4xNxM)x2 + 6x4xNxM$
- $Nbre de lectures = 6x(8xNxM)x2 + 6x(15xNxM) + 6xNxM$

Une fois, les points candidats déterminés, la prochaine étape est la suppression des non maxima qui nécessite :

- $26 comparaisons * (N - 1) * (M - 1)$ .

Après l'extraction des extrema, vient l'étape de localisation des points d'intérêt. La complexité calculatoire de cette étape est similaire à celle de l'algorithme SIFT déjà étudié dans 3.2.

- **Complexité calculatoire de l'étape calcul de l'orientation**

La première étape pour déterminer l'orientation consiste à appliquer les équations 36 et 37 à chaque point de la région qui entoure le point d'intérêt, donc 36 points (région de rayon  $6\sigma$ ). Ainsi le nombre d'opérations nécessaire pour cette première étape :

- $Nbre d'additions = (4\sigma - 1) * 4\sigma * 36 * 2 + 7*36*2$
- $Nbre de multiplications = (4\sigma * 4\sigma) * 36 * 2$
- $Nbre d'accès mémoires = (4\sigma * 2) * 4\sigma * 36 * 2 + 8 * 36 * 2$

La seconde étape consiste à calculer le vecteur  $V^T$ . (équation 38). Pour cela, il faut à chaque fois récupérer la valeur de l'angle qui est égale à :

$$angle = \arctan\left(\frac{\text{Somme des valeurs dans } ResY}{\text{Somme des valeurs dans } ResX}\right)$$

Et vérifier si cet angle se trouve dans la fenêtre. Ainsi, le nombre d'opérations nécessaire pour cette étape est :

- $Nbre d'additions = 64 * 36 + 63 = 2367$
- $Nbre de multiplications = 64 * 36 * 2 = 4608$
- $Nbre d'accès mémoires = 36 * 2 * 64 = 4608$
- $Nbre d'opérations pour l'arctangente$

Ces étapes sont calculées pour chaque point d'intérêt.

- **Complexité calculatoire de l'étape calcul du descripteur**

Pour calculer le descripteur, il faut déterminer  $\sum dx$ ,  $\sum dy$ ,  $\sum |dx|$  et  $\sum |dy|$  de chaque point d'intérêt. La première étape consiste à appliquer les équations 38 et 39 sur chaque point de la sous-région de taille  $5\sigma$  appartenant à la région qui entoure le point d'intérêt qui est de taille  $20\sigma$ . Ainsi, le nombre d'opérations nécessaire est :

- $Nbre d'additions = (6,6\sigma - 1) * 6,6\sigma * 16 * 5 * 2 + 7*16*5*2$
- $Nbre de multiplications = (6,6\sigma * 6,6\sigma) * 16 * 5 * 2$
- $Nbre d'accès mémoires = (4\sigma * 2) * 4\sigma * 16 * 5 * 2 + 8 * 16 * 5 * 2$

Enfin, la seconde et dernière étape consiste à calculer la somme des valeurs des réponses de Haar selon x et selon y ainsi que la somme des valeurs absolues des réponses de Haar selon x et selon y. Ces sommes nécessitent :

- $Nbre d'additions = 96$

### 3.6. Bilan de l'analyse en complexité de l'algorithme SURF

La phase de détection des points d'intérêt n'est pas très coûteuse en termes de nombre d'opérations. Ceci est dû à l'utilisation de l'image intégrale en conjonction avec les « boîtes filtres » et le fait de ne plus utiliser la cascade des filtres gaussiens pour construire l'espace échelle. Cependant, le fait d'utiliser l'image intégrale augmente d'une manière considérable la dynamique des valeurs à stocker en mémoire. Ainsi, on doit disposer d'assez de ressource mémoire pour pouvoir la stocker. La phase de description est la plus coûteuse en temps de calcul. Ceci est dû aux filtres gaussiens et aux filtres de Haar utilisés pour déterminer la direction dominante et calculer le descripteur.

### 3.7. Conclusion

Pour conclure ce chapitre, une comparaison des résultats obtenus lors de l'analyse de SIFT et lors de l'analyse de SURF est effectuée. La figure 28 illustre les grandes étapes menant à la construction du descripteur. Ces étapes sont les mêmes dans SIFT et dans SURF cependant les méthodes utilisées pour extraire les différents résultats sont différentes.

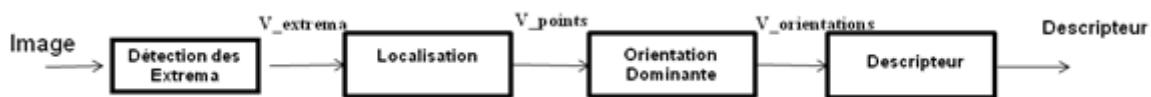


Figure 28 : Étapes menant à la construction du descripteur

Le tableau 4 contient le nombre d'opérations nécessaire pour SIFT et SURF appliqués à une image de taille 320x240 pixels en considérant 2 octaves et 4 échelles.

Tableau 4 : Comparaison de SIFT et SURF

		Détection des Extrema	Localisation	Orientation d'un point d'intérêt	Descripteur d'un point d'intérêt
Additions	SIFT	Ordre des Mop	Ordre des Kop	3 Kop	62 Kop
	SURF	Ordre des Kop	Ordre des Kop	4 Kop	8 Kop
Multiplications	SIFT	Ordre des Mop	Ordre des Kop	36 Kop	865 Kop
	SURF	-	Ordre des Kop	74 Kop	102 Kop
Divisions	SIFT	-	Ordre des Kop	6 Kop	6 Kop
	SURF	-	Ordre des Kop	-	-
Accès mémoire	SIFT	Ordre des Mop	Ordre des Kop	7 Kop	13 Kop
	SURF	Ordre des Kop	Ordre des Kop	7 Kop	174 Kop

Le tableau 4 montre que SURF présente une complexité calculatoire largement inférieure à celle de SIFT ce qui confirme les résultats obtenus dans [26].

Ainsi, suite à cette analyse, SURF offre un meilleur compromis performance qualité que SIFT. Ainsi, il va être implémenté en langage C et puis porter sur un processeur embarqué.

## 4. Implémentation de l'algorithme SURF

---

Basé sur les conclusions du chapitre précédent nous avons choisi d'implémenter SURF sur un processeur embarqué. Aucune implémentation de SURF en C n'a été trouvée. Nous allons implémenter SURF en C. Cette implémentation va être présentée dans ce chapitre.

Ce chapitre est divisé en deux parties. Une première partie dans laquelle est présentée une analyse détaillée du code C de l'algorithme SURF développé au cours du stage nommé « Z-surf ». Une seconde partie, dans laquelle est présentée une évaluation de Z-surf ainsi qu'une comparaison de Z-surf avec la version de SURF développée dans le laboratoire ETH Zurich ([www.vision.ee.ethz.ch](http://www.vision.ee.ethz.ch)).

Le code Z-surf développé doit être le plus proche possible de SURF [13], c'est-à-dire les méthodes utilisées dans SURF doivent être utilisées dans Z-surf. De plus, il doit satisfaire les critères suivants :

- Performant : Avoir un bon score de répétabilité, assurer un bon résultat d'appariement et une bonne réponse précision-rappel.
- Flexible : Avoir la possibilité de varier les paramètres de SURF d'une manière automatique.
- Possibilité de le porter sur un processeur embarqué.

### 4.1. Code Z-surf

Le code Z-surf développé représente une variante de l'algorithme SURF vu que l'expansion de taylor n'a pas été utilisée pour la localisation des points d'intérêt. D'autre part, les valeurs des échelles utilisées ne sont pas égales aux échelles utilisées dans la version originale de SURF (tableau 5). En effet, au lieu d'utiliser des échelles de types réels qui s'avèrent coûteuse en termes de complexité de calcul, nous avons décidé d'utiliser des échelles de type entier. Ceci permet d'éviter de faire à chaque fois des interpolations pour récupérer la valeur du pixel. Cette approche sera justifiée en fin de chapitre par des mesures de qualité.

Tableau 5 : Valeurs des échelles dans SURF et Z-surf

<b>Octave</b>	<b>1</b>		<b>2</b>		<b>3</b>	
<b>Niveau</b>	2	3	2	3	2	3
<b>Échelles de Z-surf</b>	2	3	4	5	7	10
<b>Échelles de SURF</b>	2	2,8	3,6	5,2	6,8	10

Le code peut être configuré en modifiant les paramètres suivants :

- Nombre d'octaves.
- Calcul en flottant ou en entier.
- Contrôle de l'affichage :
  - o Affichage ou non des points d'intérêt détectés.
  - o Affichage ou non des descripteurs.
  - o Affichage ou non de l'image intégrale.
  - o Affichage ou non de l'évolution du traitement.
- Calcul ou non de l'orientation du point d'intérêt.
- Tri ou non des valeurs des points d'intérêt détectés.
  - o Tri à bulles ou tri approximé par binning.
- Nombre de points d'intérêt à décrire.
- Mode de débogage.

La figure 29 illustre les différentes étapes du code, les entrées sorties et les liens fonctionnels.

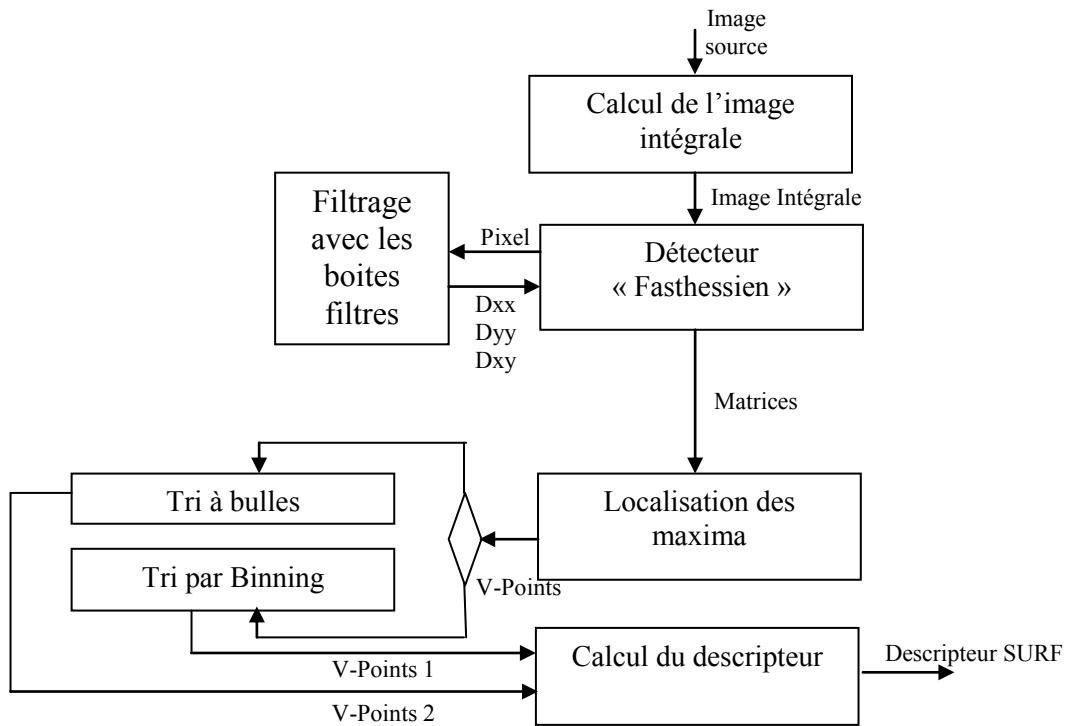


Figure 29: Organigramme de Z-surf

Z-surf est composé d'un ensemble de fonctions qui correspondent aux étapes décrites dans la figure 29. Les différentes fonctions sont :

- **Image intégrale**

Cette fonction permet de calculer à partir d'une image source l'image intégrale. En plus, si le débogage est activé elle permet de vérifier si la dynamique prévue = 32 bits a été dépassée.

- **Détecteur Fashessien**

Dans cette fonction se fait l'extraction des points candidats en calculant le déterminant de la matrice Hessienne. Ces points candidats sont situés dans des matrices qui sont le résultat de convolution des boites filtres avec l'image intégrale.

- **Filtrage avec les « boites filtres »**

Calcul le produit de convolution entre les « boites filtres » et l'image intégrale afin de générer les composantes Dxx, Dyy et Dxy de la matrice Hessienne. Cette fonction prend en entrée l'image intégrale, le type de filtre, la position du point candidat dans l'image et le « pas\_filtre » (voir figure 30).

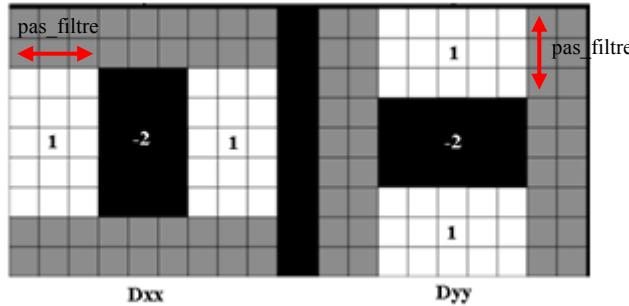


Figure 30 : « Boites filtres » de taille 9x9 selon la direction horizontale et verticale

- **Lmls**

Dans cette fonction s'effectue la suppression des non maxima et la génération du vecteur V-Points contenant l'ensemble des points d'intérêt (Figure 16).

- **Tri**

L'intérêt de trier les valeurs est de pouvoir décrire seulement les points d'intérêt les plus pertinents. Cette fonction contient deux types de tri : le tri bulle et le tri par binning. Le tri bulle permet de classer les points d'intérêt par ordre décroissant. Une fois les points d'intérêt triés nous pouvons les faire passer au descripteur en choisissant les plus grands. Ce type de tri est très coûteux en temps de calcul d'où l'intérêt du tri par binning. En effet, le tri par binning nous évite de trier toutes les valeurs en fixant un seuil qui dépend du nombre des points d'intérêt qu'on souhaite décrire.

- **Descripteur**

Cette fonction permet de générer le descripteur SURF. Elle décrit la zone qui entoure le point d'intérêt en utilisant les ondelettes de Haar en conjonction avec l'image intégrale.

## 4.2. Évaluation de Z-surf

L'évaluation de performance de Z-surf est basée sur la méthodologie proposée par Mikolajczyk [15] appliquée sur deux séquences d'images (motos et arbres). Ces deux séquences contiennent chacune 6 images prise avec une caméra dont le focus a été modifié de sorte à augmenter le flou (Annexe D). L'augmentation du flou permet de tester la réponse de Z-surf face à ce type de dégradation.

Cette évaluation est divisée en deux parties :

- Évaluation du détecteur en mesurant la répétabilité.
- Évaluation du descripteur en traçant les courbes de précision-rappel.

Les tests vont être effectués en utilisant trois octaves et en travaillant en mode « int ».

Avant de débuter l'évaluation, nous allons montrer l'effet du passage du mode « float » en mode « int ». Ensuite nous allons montrer l'effet du nombre d'octaves sur le nombre de points d'intérêt détectés.

#### 4.2.1 Passage du « float » en « int » :

L'influence du passage du mode « float » en mode « int » a été effectuée en calculant la perte d'information. Cette perte peut être déduite de la figure 32 en faisant la soustraction entre le nombre de points détectés en mode « float » avec ceux détectés en mode « int ». La perte la plus significative s'est produite pendant la première octave. En effet, 2500 points n'ont pas été détectés. Cependant, ces pertes n'ont pas une grande importance, puisque elles n'ont pas eu un impact important sur la répétabilité (figure 31) où seulement une perte de 2% au niveau de toutes les octaves a été notée.

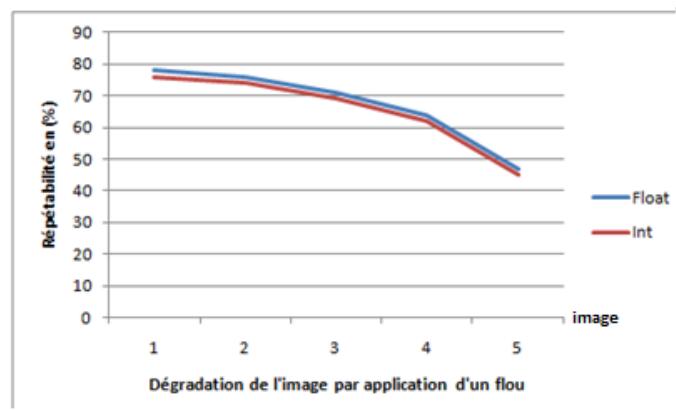


Figure 31 : Effet du passage du « float » en « int » sur la répétabilité

La figure 32 représente le nombre de points d'intérêt détecté en fonction du nombre d'octaves. Elle montre que la quatrième et la cinquième octave n'ont pas d'influence sur le nombre de points d'intérêt détectés. Ce résultat confirme ce qui a été dit dans l'article d'Herbert Bay [13], à savoir qu'il n'est pas nécessaire de calculer plus que 3 octaves.

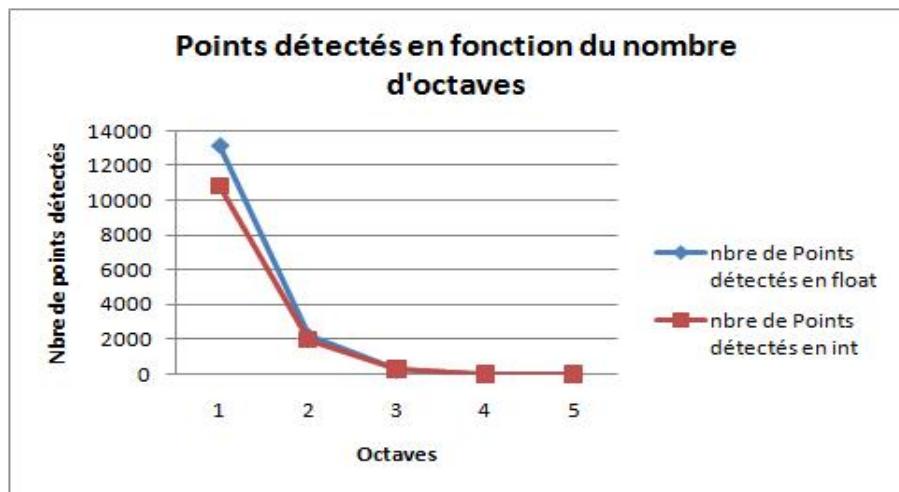


Figure 32 : Nombre de points d'intérêt détectés en fonction du nombre d'octaves

#### 4.2.2 Évaluation du détecteur

Le critère le plus utilisé pour évaluer la performance d'un détecteur est la répétabilité. Les figures 33 et 34 montrent la répétabilité de deux implémentations Z-surf et SURF appliquées respectivement sur les séquences d'images de l'annexe D de sorte que Z-surf et SURF détectent 3000 points d'intérêt.

Pour les deux séquences, Z-surf atteint un meilleur résultat que SURF. En effet, pour la séquence de motos, Z-surf atteint le score de 76% alors que SURF atteint le score de 65% au niveau de la première dégradation. Pour la séquence d'arbres, Z-surf atteint un score moins bon que le score de la moto (60% au niveau de la première dégradation) mais il reste encore meilleur que SURF (56%). La séquence d'arbres contient plus de détails que la séquence de motos de plus les zones sont presque similaire d'où la chute de 20% du score de répétabilité.

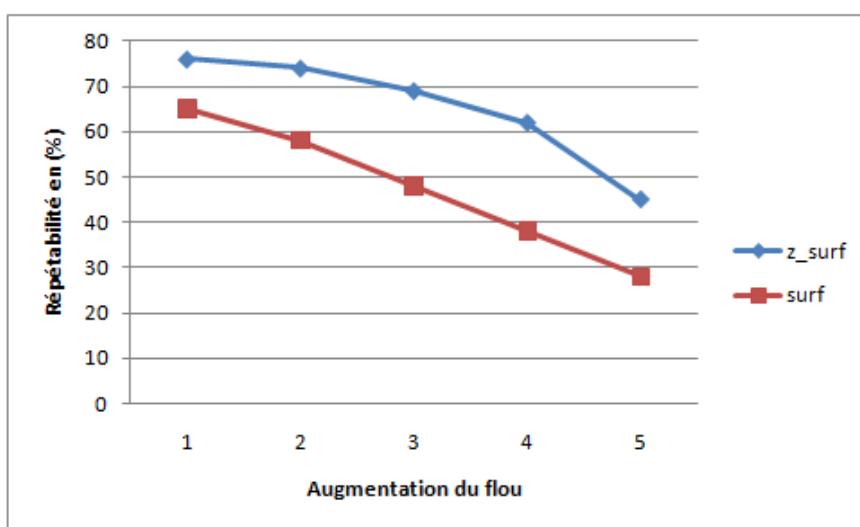


Figure 33 : Variation de la répétabilité en fonction du flou pour la séquence de motos

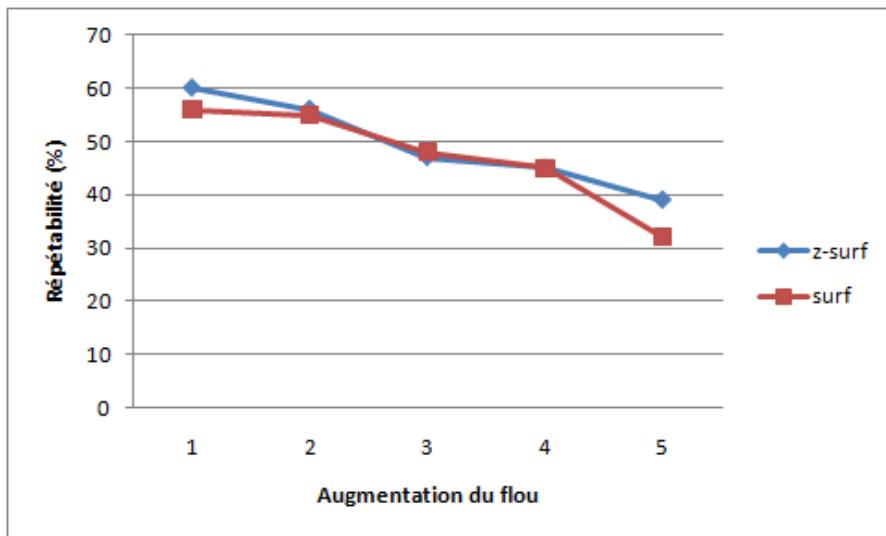


Figure 34 : Variation de la répétabilité en fonction du flou pour la séquence d'arbres

#### 4.2.3. Évaluation du descripteur

L'évaluation des descripteurs se base sur les courbes de précision-rappel. Les figures 35 et 36 montrent respectivement le résultat d'évaluation de deux implémentations Z-surf et SURF appliquées aux séquences d'images 3a et 3b. Z-surf et SURF ont des résultats globalement assez proche. La figure 35 montre que Z-surf a un meilleur résultat que SURF. En effet, pour un rappel égal à 20% Z-surf possède une précision de 50% alors que SURF possède une précision de 40%. La figure 36 montre que SURF et Z-surf ont des résultats assez proches bien que la séquence d'images 3b contient beaucoup de détails. Ces deux exemples permettent de valider l'implémentation de Z-surf vu que nous obtenons des résultats satisfaisants.

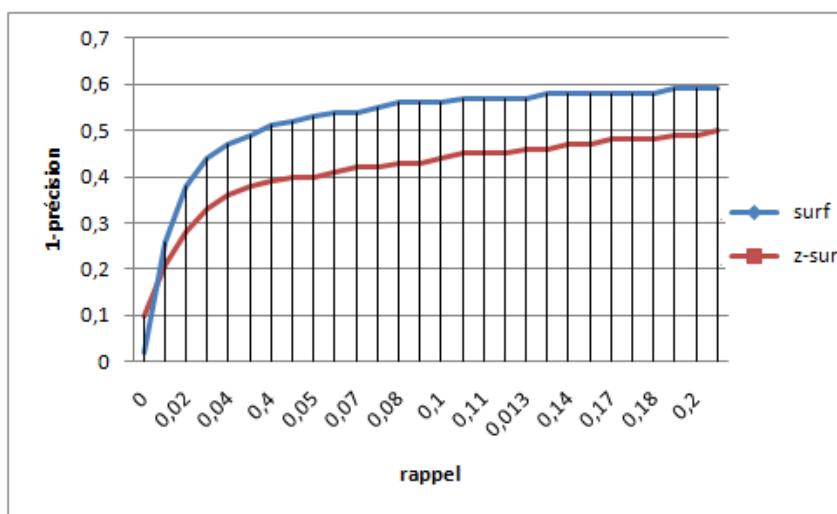


Figure 35 : Différence entre Z-surf et SURF (figure 3a motos)

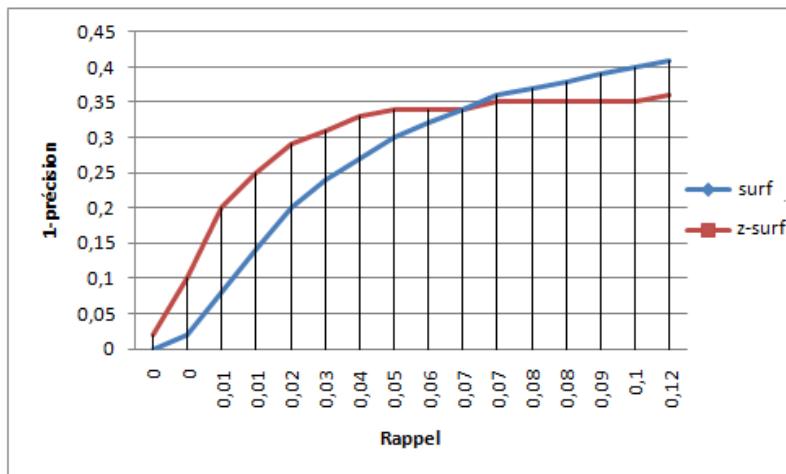


Figure 36 : Différence entre Z-surf et SURF (figure 3b arbres)

#### 4.3. Profiling de Z-surf

Le profiling de Z-surf est obtenu en utilisant « gprof ». Pour avoir des résultats précis, nous avons répété 20 fois l'exécution du même code. Cette répétition, explique le temps de traitement important illustré dans la figure 37. Cette figure montre l'influence du nombre d'octaves sur le temps de calcul. Nous pouvons constater que le temps de calcul augmente si nous augmentons le nombre d'octaves. Cependant, au-delà de trois octaves le temps n'augmente plus. Ceci vient du fait que pendant la quatrième et la cinquième octave presque aucun points d'intérêt ne sont détectés.

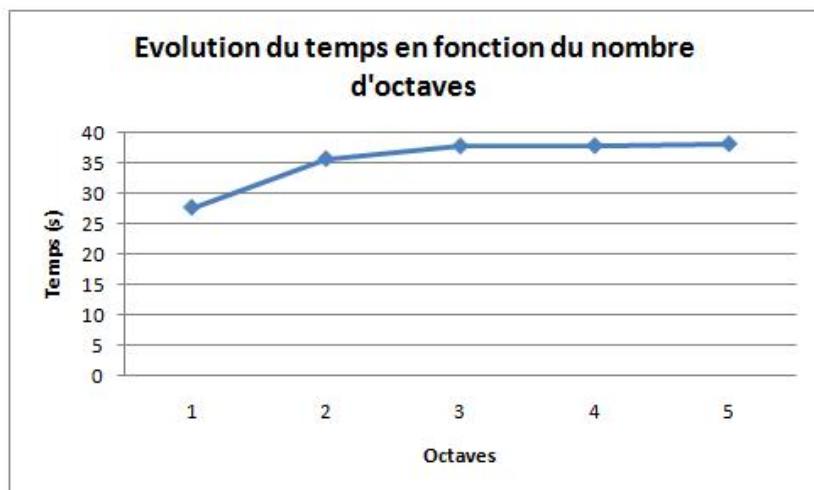


Figure 37 : Temps en secondes en fonction du nombre d'octaves

Gprof permet aussi de connaître le pourcentage d'occupation processeur de chaque fonction. Ainsi, des optimisations peuvent être effectuées pour les fonctions qui occupent le plus le processeur. Le tableau 6 montre que la fonction Tri bulle est la plus coûteuse. Ainsi, le tri bulle a été remplacé par le tri par binning ce qui a eu pour conséquence de diminuer considérablement le temps de calcul de la phase Tri.

Tableau 6 : Temps processeurs occupé de chaque fonction pour l'octave 3 cas du tri bulle

	Image intégrale	Fasthessian	Lmls	Tri à bulles	Descripteur
Temps d'occupation du processeur (%)	0,18	10,52	3,89	59,14	25,15

Tableau 7 : Temps processeurs occupé de chaque fonction pour l'octave cas du tri par binning

	Image intégrale	Fasthessian	Lmls	Tri par binning	Descripteur
Temps d'occupation du processeur (%)	0,27	29,05	10,39	0,09	56,74

Le tableau 7 montre que le temps d'occupation du tri a largement diminué. Maintenant, c'est la phase descripteur qui est la plus coûteuse en temps de calcul. Ceci est dû au grand nombre d'appels des fonctions qui permettent de calculer la réponse de Haar selon x et selon y en chaque point de la fenêtre de taille  $20\sigma$  autour du point d'intérêt.

Tableau 8 : Profiling des sous-fonctions de « Fashessian » et de « Descripteur »

		Temps d'occupation du processeur (%)/Z_surf	Temps d'occupation du processeur (%)/la fonction	Nombre d'appels
Fasthessian	Cal_det_i	6,04	6,04	20,82
	Filtre_xx	(6,94)	23,93	59540080
	Filtre_yy	(6,49)	22,37	59540080
	Filtre_xy	(9,53)	32,89	59540080
Descripteur	Box_integral	32,78	57,77	916960000
	Resp_Haar_xx	5,9	10,39	229240000
	Resp_Haar_xx	8,76	15,43	229240000
	Descripteur	9,3	16,39	20

Le tableau 8 montre que la fonction Fasthessian occupe 29% du traitement alors que le descripteur occupe 56% du traitement. Cette différence est due au grand nombre d'appels de la fonction Box\_integral qui est appelée 916960000. Le traitement effectué dans Box\_integral est à-peu-près similaire à celui effectué dans filtre\_xx, filtre\_yy et filtre\_xy qui est un simple calcul d'image intégrale (les fonctions filtres = 22,96% et Box\_integral =32,78). Ainsi, au début nous allons porter la phase de détection dans un processeur embarqué en optimisant les fonctions des boites filtres (filtre\_xx, filtre\_yy et filtre\_xy). Ces mêmes optimisations pourront être utilisées pour la fonction Box\_integral.

#### 4.4. Conclusion

Le code Z-surf développé est efficace puisqu'il a une bonne répétabilité, un bon résultat d'appariement et il est robuste aux dégradations par application de flou. En effet, on obtient des résultats à-peu-près égaux à SURF. Nous avons une répétabilité en moyenne égale à 70% et des courbes de précision-rappel dont le résultat est satisfaisant (pour la séquence moto image 3a nous obtenons un rappel égal à 20% pour une précision de 50%). De plus, il a la caractéristique d'être flexible, vu que le passage du mode float en mode int n'altère pas considérablement la performance du code. En effet, ce passage fait chuter le score de répétabilité de 2%. D'autre part, il est possible de varier plusieurs paramètres : le nombre d'octaves, le nombre de points que nous souhaitons décrire, l'affichage...

Le profiling effectué du code Z-surf a permis de dégager les fonctions les plus coûteuses en temps de calcul qui sont la fonction Box\_integral (32%) et les fonctions filtre\_xx, filtre\_yy et filtre\_xy (23%). Le traitement dans ces fonctions est à-peu-près similaire, donc avant de porter tout le code et optimiser l'ensemble, nous avons décidé de porter sur processeur embarqué seulement la phase de détection et optimiser cette phase, car l'optimisation de cette phase passe par l'optimisation des fonctions filtre\_xx, filtre\_yy et filtre\_xy donc d'une manière indirecte optimiser la phase de description car on optimise la fonction Box\_integral.

## 5. Portage de Z-surf sur AntX

Ce chapitre présente les différentes phases de portage sur AntX de la première partie de Z-surf à une octave et quatre échelles. Il est divisé en trois parties : une première partie dans laquelle est présenté le processeur Antx sur lequel a été porté Z-surf. La deuxième partie, dans laquelle sont définies les modifications apportées à Z-surf de sorte à l'adapter aux contraintes matérielles. La troisième partie dans laquelle est exposé le résultat de portage.

### 5.1. Le processeur AntX

Le processeur de contrôle AntX [25] développé au sein du Laboratoire de Calcul Embarqué (LCE) est un processeur de type RISC (Reduced Instruction Set Computer). Ce type d'architecture présente l'avantage d'être bien connu, tout comme ses outils de programmation.

Le jeu d'instructions est RISC, et possède des instructions 16 et 32 bits. Ces instructions sont générées grâce au compilateur associé au processeur et développé lui aussi au sein du laboratoire. Il est basé sur le compilateur GCC et supporte donc toutes les fonctions de ce dernier tel que le désassemblage de l'exécutable produit. En outre, un script python permet, à partir du fichier désassemblé, d'extraire les mémoires de données et d'instructions servant à alimenter le processeur.

Le processeur est constitué de 5 étages illustrés à la figure 38.

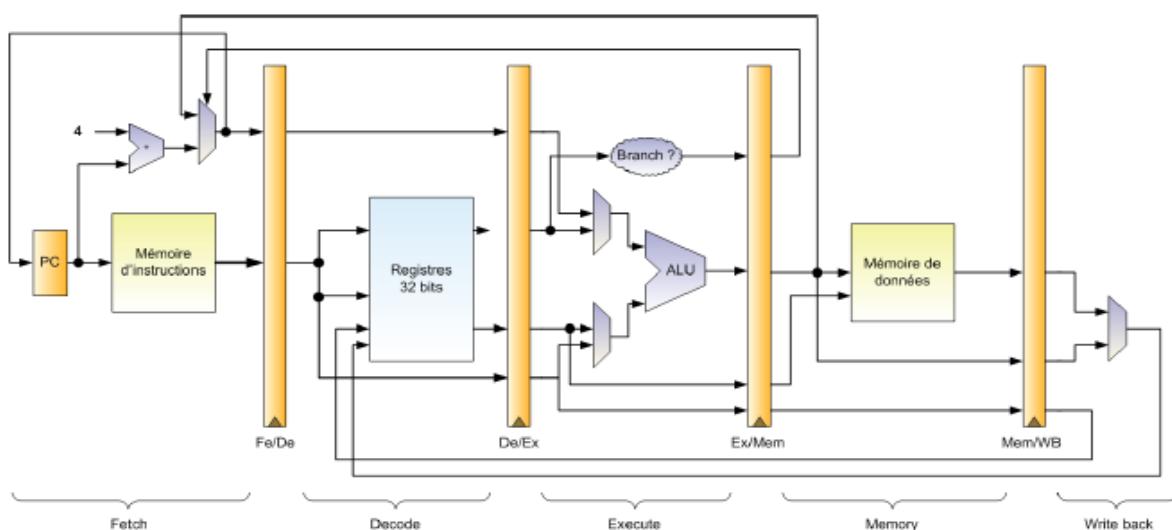


Figure 38 : Pipeline du processeur de contrôle constitué de 5 étages avec 16 registres généraux de 32 bits

L'étage de fetch gère le réalignement des instructions 16 et 32 bits en provenance de la mémoire d'instruction. L'étage data offre au développeur une interface mémoire afin

d'augmenter la taille de la mémoire si nécessaire. Cet étage offre de plus une interface coprocesseur afin de spécifier des instructions supplémentaires.

## 5.2. Adaptation de Z-surf aux contraintes matérielles

L'objectif du portage de Z-surf sur AntX est de vérifier son bon fonctionnement avant de le porter sur une architecture distribuée à base de processeurs de type AntX. D'autre part, ce portage donne accès à un profiling précis de Z-surf.

Dans le cadre d'une architecture distribuée, chaque processeur AntX traite un bloc de l'image et non l'image entière.

Une image de taille 64x64 a été choisie pour le portage sur AntX. Cette taille permet d'avoir assez de pixels à traiter compte tenu des effets de bord dû à la taille 27x27 du « boîte filtre » utilisé à l'échelle 4 de la première octave.

Dans un premier temps il est nécessaire de déterminer la quantité mémoire nécessaire au portage.

Ainsi, pour détecter les points d'intérêt d'une image de taille 64x64 en utilisant Z-surf. Il faut tout d'abord :

- Stocker l'image source de taille 64x64 = 4096 mots.
- Stocker l'image intégrale de taille 64x64 = 4096 mots.
- Stocker les 4 matrices de taille 64x64x4 = 16384 mots.
- Stocker les points d'intérêt détectés : en limitant par un seuillage de 300 le nombre de points détectés donc :  $300 \times 5 = 1500$  mots (dans un point d'intérêt nous avons les coordonnées (x,y) du point, l'échelle, la valeur du déterminant et la direction dominante).

Ici un mot correspond à une donnée de 32 bits ; Ainsi pour une image de taille 64x64 un besoin de mémoire égale à  $16384 + 4096 + 4096 + 1500 = 26076$  mots = 102 Ko est nécessaire ce qui correspond à une surface égale à  $2\ 040\ 000\ \mu\text{m}^2$  en technologie 90 nm.

L'image source est utilisée seulement pour le calcul de l'image intégrale. Ainsi, l'espace alloué à l'image source peut être utilisé pour y stocker l'image intégrale et ainsi un gain de 16 Ko est obtenu. La grande majorité de l'espace mémoire est utilisé pour stocker les 4 matrices dans lesquelles la suppression des points non-maxima est effectuée. Ainsi, pour réduire la mémoire il faut réduire la taille de ces matrices. Une idée consisterait à vérifier si un pixel est un point d'intérêt lors de son traitement et non pas après avoir calculé les 4 matrices. Ainsi, le stockage des valeurs des déterminants de chaque point de l'image dans les 4 matrices n'est plus indispensable, mais seulement le stockage des 26 voisins qui entourent le point considéré ainsi que le point considéré (voir figure 16).

Donc, la quantité de mémoire nécessaire devient :

- image intégrale : 4096 mots
- les 4 déterminants :  $9 \times 4 = 36$  mots
- les points d'intérêts : 1500 mots

Pour une image de taille 64x64 un besoin de mémoire égale à  $4096+1500+36 = 5632$  mots = 22 Ko sont nécessaires ce qui correspond à une surface égale à  $440\ 000\ \mu\text{m}^2$  en technologie 90 nm.

La figure 39 représente l'organigramme de Z-surf pour la phase de détection des points d'intérêt ainsi que l'ordre d'appel de chaque fonction (les nombres en rouge). La différence avec l'organigramme de la figure 29 est qu'ici le traitement s'effectue pixel à pixel.

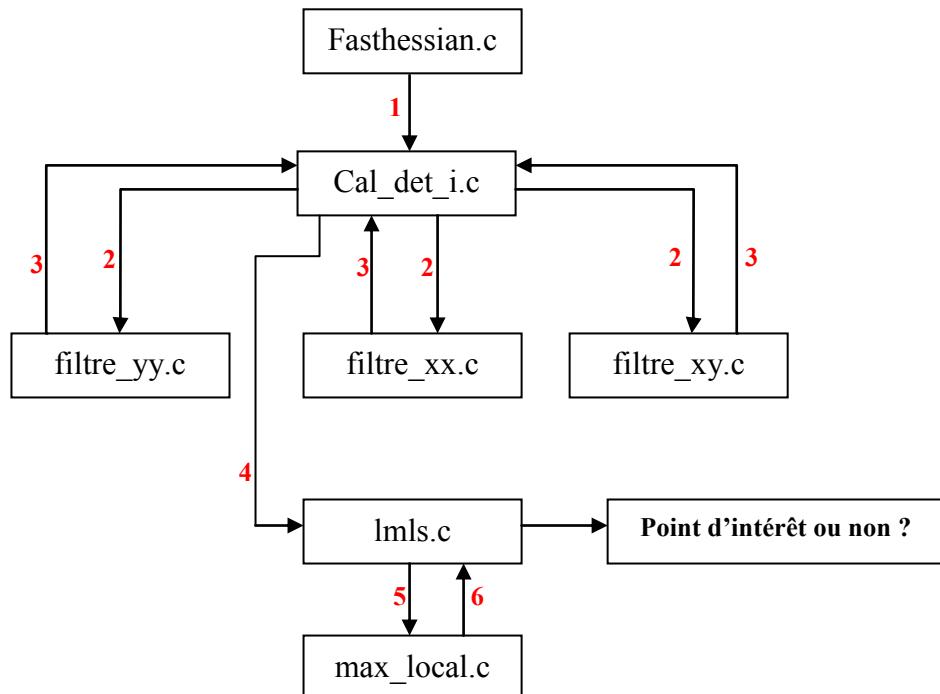
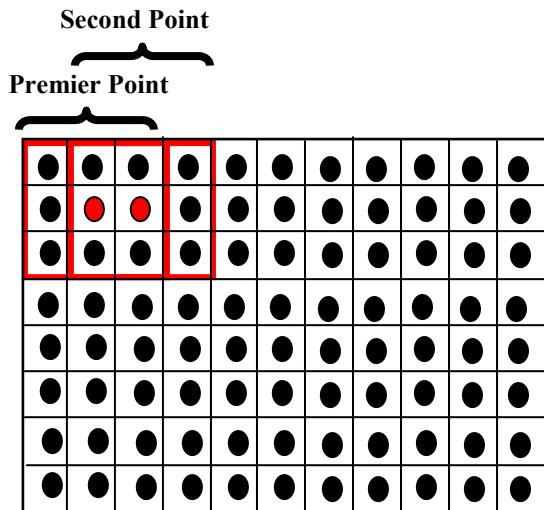


Figure 39 : Nouvel organigramme de Z-surf

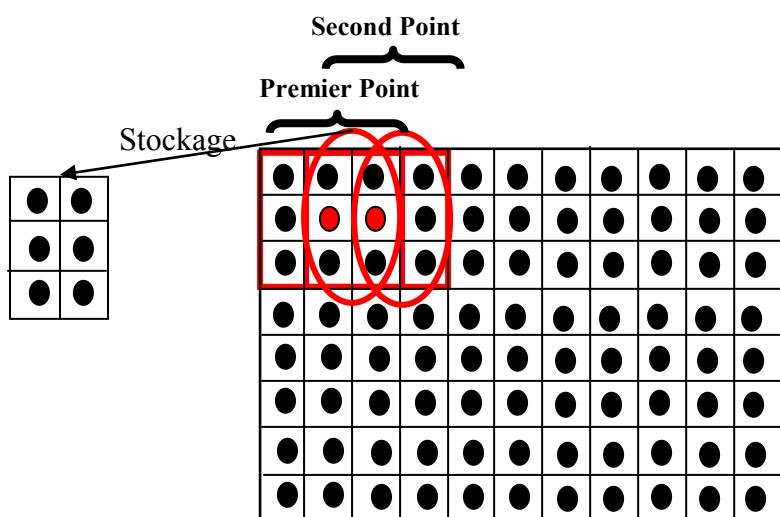
L'inconvénient de la seconde approche est l'augmentation du nombre d'opérations. En effet, pendant la phase de détection les fonctions `filtre_xx`, `filtre_yy` et `filtre_xy` n'étaient appelées qu'une seule fois pour chaque point de l'image afin de déterminer si un pixel est un point d'intérêt. Dans cette seconde approche, ces fonctions sont appelées chacune 9 fois pour un point donné de l'image alors qu'avant elles n'étaient appelées qu'une seule fois. En effet, dans cette nouvelle version on calcule les déterminants des 9 voisins qui entourent le point candidat. Ensuite, on vérifie s'il est extrema ou non et on répète cette opération pour chaque point de l'image. De ce fait, 6471689 additions sont nécessaires pour extraire les points d'intérêt d'une image de taille 64x64 alors qu'avec l'ancienne version de Z-surf il nous fallait 679936 additions. Ainsi la deuxième version permet de diminuer la mémoire de 78,5% ce qui a pour effet de réduire la surface de la mémoire de 78,5% aussi en technologie 90 nm mais elle met en jeu environ 10 fois plus de calcul. La figure 40 montre la démarche employée dans la deuxième approche pour extraire les points d'intérêt.



**Image intégrale**

Figure 40 : Deuxième approche pour l'extraction des points d'intérêt

Une troisième approche réalisant un compromis mémoire/nombre d'opérations peut être employée. Cette approche consiste à stocker les déterminants des pixels qui vont être utilisés pour vérifier si le point de l'image juste à droite du pixel considéré est extrema ou non (figure 41). Cette approche met en jeu 3 fois plus de calcul par rapport à la version initiale. La figure 41 montre la démarche employée dans la troisième approche pour extraire les points d'intérêt. Ainsi, 6x4 cases mémoires supplémentaire par rapport à l'autre approche sont utilisées. Donc, pour une image de taille 64x64 un besoin en mémoire égale à  $5632 + 6 \times 4$  mots = 5656 mots = 22,09 Ko est nécessaire ce qui correspond à une surface de  $441\ 800\ \mu\text{m}^2$ .



**Image intégrale**

Figure 41 : Troisième approche pour l'extraction des points d'intérêt

Une quatrième approche réalisant aussi un compromis mémoire/nombre d'opérations peut être employée. Cette approche consiste à stocker à chaque fois trois lignes de l'image intégrale (figure 42). L'avantage de cette approche est qu'il n'y a pas de calcul supplémentaire par rapport à la version initiale. Pour une image de taille 64x64 un besoin en mémoire égale à  $5632+64 \times 3 \times 4 = 6400$  mots=25Ko est nécessaire ce qui correspond à une surface de 500 000  $\mu\text{m}^2$ . La figure 42 montre la démarche employée dans la quatrième approche pour extraire les points d'intérêt.

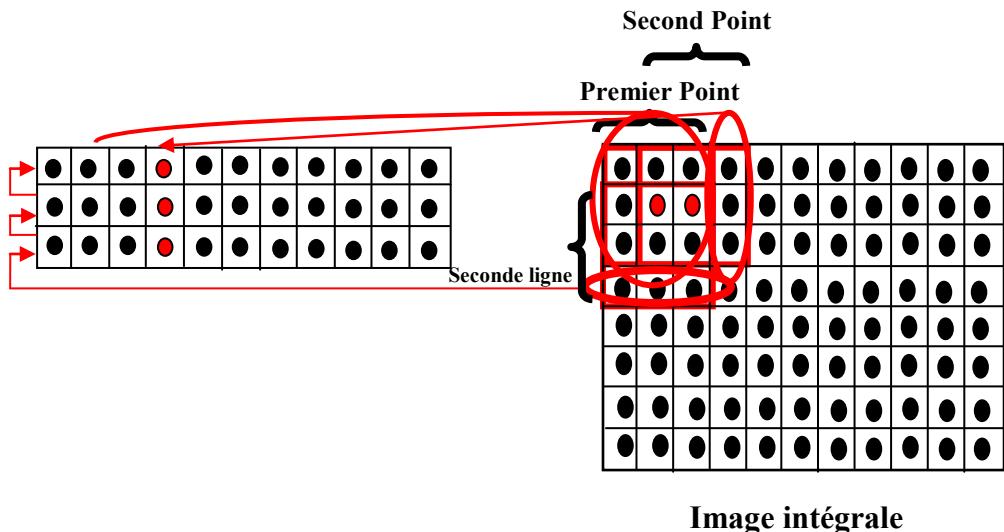


Figure 42 : Quatrième approche pour l'extraction des points d'intérêt

### 5.3. Portage de la deuxième approche de Z-surf sur AntX

Dans un premier temps, seulement le portage de phase de détection de la nouvelle version de Z-surf à une octave sera effectué.

Avant de porter Z-surf sur Antx, les zones mémoires occupées par l'image intégrale, les déterminants et les points d'intérêt doivent être spécifiées. Une mémoire de taille 32 Ko (8192 mots) va être utilisée. La figure 43 illustre le début d'adresse de chaque composante dans le mémoire de 32 Ko :

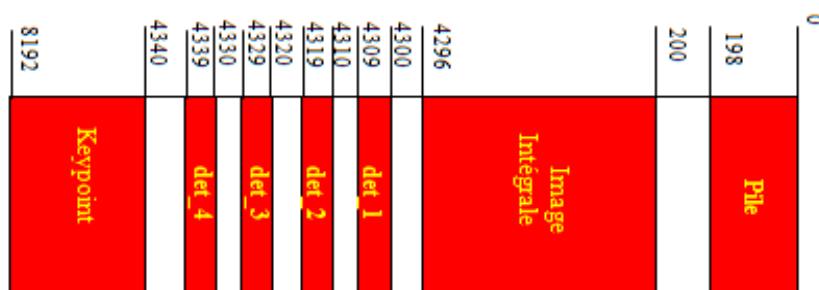


Figure 43 : zone mémoire occupée pour chaque composante

Une fois que les paramètres de Z-surf ont été définis, l'extraction des résultats de profiling se fait automatiquement de la façon suivante : le code Z-surf est compilé. Cette compilation génère un fichier main.S contenant les adresses de début et de fin de chaque fonction. Un script Python récupère d'une manière automatique ces adresses et les fait passer à Modelsim sous la forme d'un script Tcl. La simulation comportementale est alors lancée de manière interactive ou console, cette dernière présentant l'avantage d'être plus rapide mais ne permet pas de débugger le processeur. Le script Tcl se charge alors à chaque coup d'horloge de calculer le nombre de lectures, d'écritures, de cycles et d'appels de chaque fonction et renvoie après le résultat à la console.

Le tableau 9 contient le résultat de profiling de Z-surf appliqué sur une image de taille 64x64 :

Tableau 9 : Résultats de profiling

	Nbre d'appels	Nbre de cycles	Nbre de lectures	Nbre d'écritures	Nbre de cycles par appel	Nbre de lectures par appel	Nbre d'écritures par appel	% Relatif total
filtre_xx	39204	2901096	392040	117612	74	10	3	29,64
filtre_yy	39204	2450250	378972	120161	63	10	3	25,03
filtre_xy	39204	4173157	771012	124154	106	20	3	42,64
max_local	2178	37026	8712	6534	17	4	3	0,38
lmls	2178	154638	39204	4356	71	18	2	1,58
Image intégrale	1	70028	8130	4098	70028	8130	4098	0,71
main	1	18	1	3	1	1	3	0
Total	121970	9786213	1598071	376918	-	-	-	100

Ce profiling sur un processeur embarqué confirme que la phase de détection des points candidats par l'intermédiaire des fonctions filtre\_xx, filtre\_yy et filtre\_xy occupe la majeure partie du traitement. Ainsi pour diminuer la charge de calcul du processeur il serait envisageable de mettre en place des coprocesseurs adaptés à chacun de ces calculs.

#### 5.4. Conception de l'interface Coprocesseur

Dans cette partie, nous allons détailler la conception du coprocesseur. Ce coprocesseur permet de calculer les adresses des données utilisées pour calculer le produit de convolution entre le « boîte filtre » et l'image intégrale. Une fois les adresses calculées, les données sont récupérées et le produit de convolution est calculé. Le résultat de convolution est ensuite envoyé par le coprocesseur vers le processeur AntX.

Ci-dessous, le code C de la fonction filtre\_xx qui permet de calculer la composante Dxx de la matrice Hessienne.

```

int filtre_xx(int **image_integrale, int filtre, int row, int col)
{
    int A,B,C,D,E,F,G,I,x1,x2,x3,x4,y1,y2,b,l;
    switch (filtre) {

        case 9 :
            b=4; l=3; rows=5;
            break;

        case 15 :
            b=7; l=5; rows=9;
            break;

        case 21 :
            b=10; l=7; rows=13;
            break;

        case 27 :
            b=13; l=9; rows=17;
            break;

        y1=row-1;
        y2=y1+rows;
        x1=col-b-1;
        x2=x1+l;
        x3=x2+l;
        x4=x3+l;
        A=image_integrale[y1][x1];
        B=image_integrale[y1][x2];
        C=image_integrale[y2][x1];
        D=image_integrale[y2][x2];
        E=image_integrale[y1][x3];
        F=image_integrale[y1][x4];
        G=image_integrale[y2][x3];
        I=image_integrale[y2][x4];
    }

    return -A-3*D+3*B+C+3*G-3*E-I+F;
}

```

Récupération des paramètres de la ROM

Calcul des adresses

Lecture des données de la RAM

Envoi du résultat à AntX

### Listing du code source de la fonction filtre\_xx

Le coprocesseur va se charger d'exécuter les différentes instructions de la fonction filtre\_xx. Il est constitué de 2 unités arithmétiques et logiques (ALU), d'un séquenceur, de 12 registres, de 2 multiplexeurs et d'un additionneur multiplicateur (voir figure 45). Les paramètres b, l et rows dans filtre\_xx sont constants pour un type de filtre donné. Ainsi, leurs valeurs sont stockées en ROM. Par contre, l'image intégrale est variable et dépend de l'image en entrée ainsi elle va être stockée en RAM. La première ALU sert à calculer les adresses d'accès aux 8 points de l'image intégrale (A, B, C, D, E, F, G et I). La deuxième ALU est utilisée pour calculer le produit de convolution entre la boîte filtre et l'image intégrale c'est-à-dire :  $-A-3D+3B+C+3G-3E-I+F$ . Les registres sont utilisés pour stocker les valeurs intermédiaires entre les différents traitements. Les 2 multiplexeurs permettent de concentrer sur le bus de données une seule entrée parmi les 8 possibles. Le séquenceur permet de coordonner le traitement effectué dans le coprocesseur et de calculer le produit de convolution entre les boîtes filtres et l'image intégrale. C'est lui qui active et désactive les différents blocs du coprocesseur c'est-à-dire les ALUs, les MUXs et les regisres. La figure 44 représente les différentes étapes effectuées par le séquenceur. Tout d'abord, b, l et rows sont récupérés de la ROM et envoyé à l'ALU. Ensuite, l'ALU avec ces paramètres et les coordonnées du point (row,col) génère les

positions  $(x_i, y_i)$ . Ces positions vont être ensuite stockées dans 8 registres. Une fois ces positions stockées dans les registres les 2 multiplexeurs sont activés et permettent d'envoyer à l'additionneur multiplicateur les positions  $x_i$  et  $y_i$  nécessaires pour déterminer les adresses de A, B, C, D, E, F et I dans la RAM. Pendant ce temps, la deuxième ALU est activée. Cette ALU effectue les 8 opérations nécessaire pour déterminer le résultat du produit de convolution entre le boîte filtre et l'image intégrale. Ainsi, elle prend en entrée la donnée récupérée de la RAM si le bus de données est libre (si le bus de données n'est pas libre elle attend) et la valeur de donnée déjà calculée stockée dans un registre. Une fois les 8 opérations effectuées le résultat est envoyé du coprocesseur vers le processeur AntX en activant le registre situé au dessous de l'ALU.

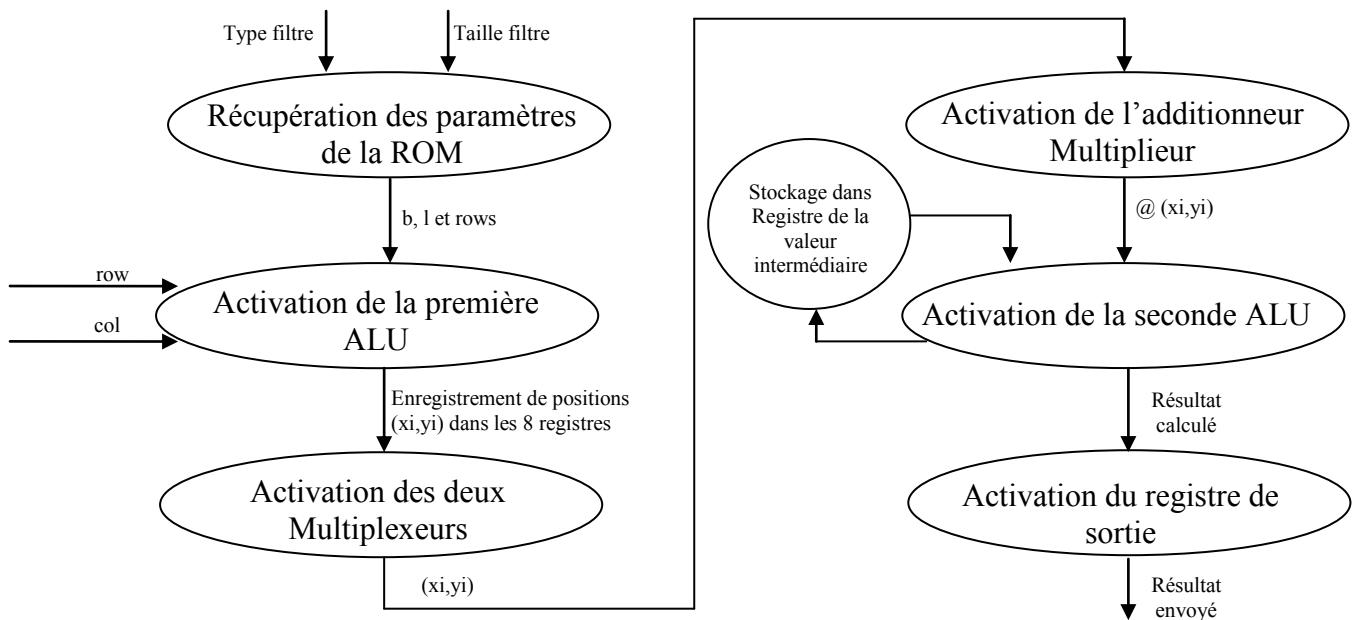


Figure 44 : Organigramme du séquenceur

Type filtre Taille filtre

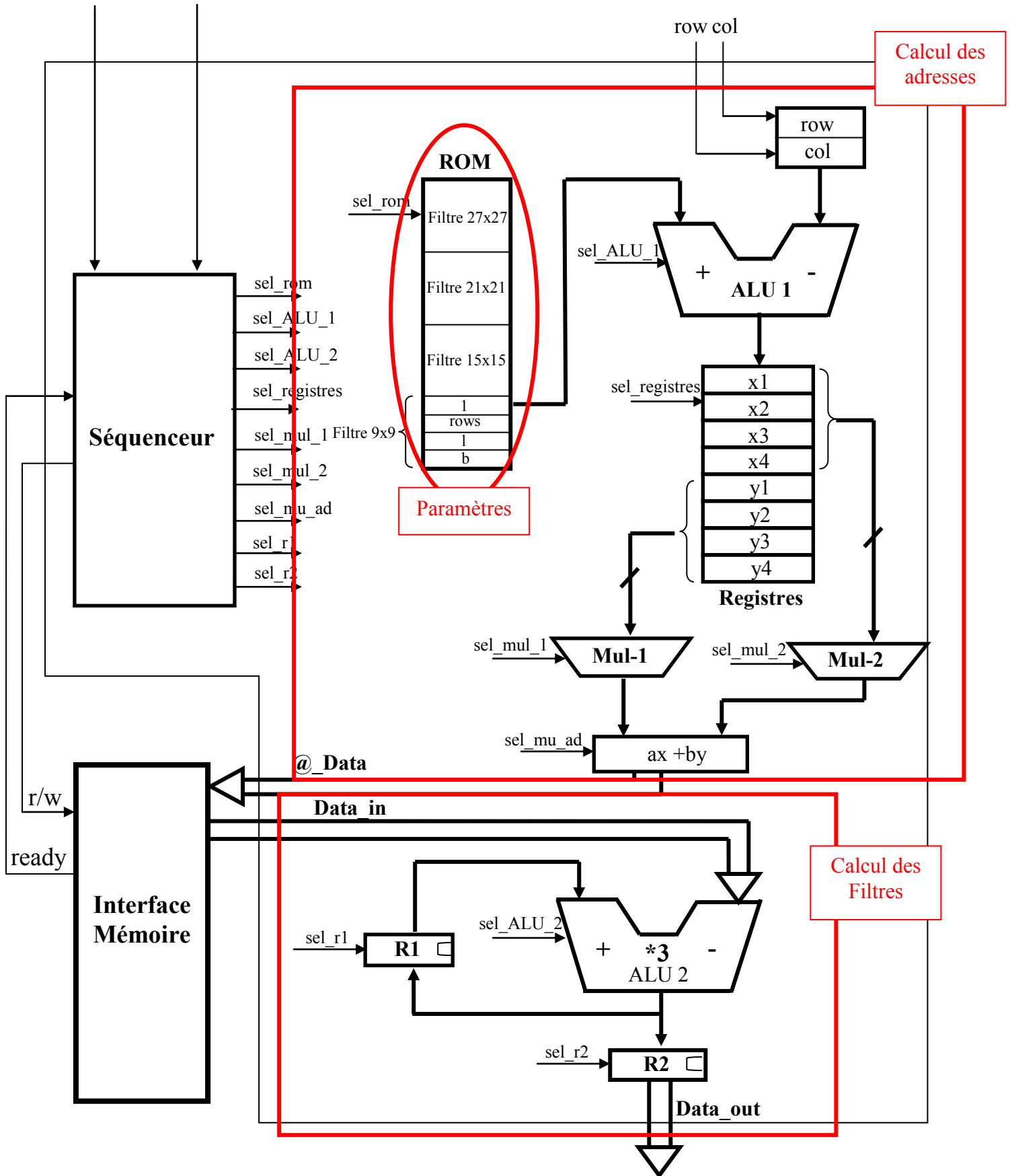


Figure 45 : Architecture du Coprocesseur

Le tableau 10 montre les opérations effectuées dans le coprocesseur à chaque cycle d'horloge.

Tableau 10 : Nombre de cycles nécessaires pour envoyer le résultat à AntX

Cycles	Opérations
1	Calcul de y1
2	Calcul de x1
3	Calcul de x2
4	Chargement de : B ; Calcul de y2
5	Chargement de : D ; Calcul de : x3, (+B)
6	Chargement de : E ; Calcul de : x4, (+B-D)
7	Chargement de : G ; Calcul de : (+B-D-E)
8	Calcul de : (B-D-E+G)
9	Chargement de : A ; Calcul de : 3* (B-D-E+G)
10	Chargement de : I ; Calcul de : 3* (B-D-E+G)-A
11	Chargement de : F ; Calcul de : 3* (B-D-E+G)-A-I
12	Chargement de : C ; Calcul de : 3* (B-D-E+G)-A-I+F
13	Chargement de : C ; Calcul de : 3* (B-D-E+G)-A-I+F+C
14	Envoi du résultat

Le tableau 10 montre que le nombre de cycles total nécessaire pour calculer le produit de convolution est de 14 cycles. En comparant ce résultat avec celui présenté dans le tableau 9, un gain de 81,08% a été enregistré. En effet, pour calculer le produit de convolution entre le « boite filtre » selon la direction horizontale et l'image intégrale, 74 cycles étaient nécessaire alors que maintenant seulement 14 cycles sont nécessaires.

## 5.5. Conclusion

Il a été évoqué dans ce chapitre les différentes étapes menées pour porter Z-surf sur AntX. Une fois l'algorithme porté dans le processeur, nous avons dégagé le nombre de cycles nécessaires ainsi que le nombre de lectures et écritures effectuées dans chaque fonction. Il s'est avéré que la phase de détection des points candidats est la plus coûteuse. En effet, elle représente 96,71% du traitement.

Ainsi, dans le but d'optimiser le temps d'exécution, un coprocesseur a été conçu. Ce coprocesseur est responsable du traitement effectué dans les fonctions filtre\_xx, filtre\_yy et filtre\_xy et il a permis de diminuer de 81,08% le nombre de cycles dans la fonction filtre\_xx. La perspective de ce portage consiste à utiliser une architecture distribuée d'AntX de sorte que chaque AntX traite une partie de l'image. Ainsi, il sera possible de traiter des images de grande taille et de paralléliser le calcul donc faciliter le passage au temps réel. Cependant, l'utilisation d'une architecture distribuée conduit à utiliser une image intégrale distribuée qui peut générer des « overheads » de calcul important. D'autre part, les données peuvent être éloignées ce qui augmente leur temps de récupération, donc le temps d'exécution.

# Conclusion Générale

---

Ce stage de fin d'études effectué au sein du Laboratoire de Calcul Embarqué du CEA LIST m'a permis d'intégrer une équipe pluridisciplinaire.

Une étude bibliographique menée lors du stage a permis de faire un état de l'art des différents algorithmes de détection et de description des points d'intérêt. Grâce à cet état de l'art nous avons pu déterminer que SIFT et SURF sont les algorithmes les plus performants. Ainsi, une analyse plus approfondie de ces deux algorithmes a été effectuée dans le but de dégager lequel est le plus performant. Cette analyse a permis de déduire que SURF assure le meilleur compromis performance/complexité. En effet, le tableau 4 montre que SURF présente une complexité calculatoire largement inférieure à celle de SIFT. De plus, le tableau 2 et la figure 14 extraits de [20] montrent que SURF est plus performant que SIFT et que son temps d'exécution est inférieur de 24% à celui de SIFT.

La prochaine étape consistait à coder l'algorithme SURF en langage C avec format de données flottant tout en mettant en place une méthodologie de comparaison des résultats obtenus avec la librairie de l'algorithme commercial SURF. Cette méthodologie se basait sur la mesure de la répétabilité pour l'évaluation des détecteurs et les courbes précision-rappel pour l'évaluation des descripteurs.

La dernière étape, consistait à adapter le code pour le porter sur le processeur AntX. Une étape très importante a alors consisté à transformer cet algorithme avec un encodage des données à virgule fixe tout en maîtrisant la dégradation des performances. En effet, cette transformation n'a causé qu'une chute de 2% sur le score de répétabilité. Le portage de ce code a permis d'effectuer un profiling précis. Ce profiling nous donne une idée sur les fonctions qui sont les plus coûteuses en temps de calcul. Ces fonctions sont celles qui effectuent le produit de convolution entre les boîtes filtres et l'image intégrale (filtre\_xx, filtre\_yy, filtre\_xy et Box\_integral). En effet, elles représentent 54% du temps d'exécution (22% pour la phase de détection et 32% pour la phase de description).

La phase de détection a été portée sur le processeur AntX. Ce portage a permis de confirmer que les fonctions coûteuses sont les fonctions filtre\_xx, filtre\_yy et filtre\_xy qui représente 96,71% du temps d'exécution. Ainsi, dans le but d'optimiser la phase de détection de Z-surf, un coprocesseur a été conçu. Ce coprocesseur est destiné à réaliser le traitement effectué dans ces fonctions. Il a permis de diminuer le nombre de cycles de 81,08%.

Comme perspectives, nous projetons, en premier lieu, d'optimiser la nouvelle version de Z-surf portée sur AntX en faisant un compromis entre la mémoire et le nombre d'opérations. En second lieu, nous prévoyons de porter Z-surf entier et vérifier son bon fonctionnement par rapport à la version originale de SURF. Ensuite, nous envisageons de réaliser le coprocesseur présenté dans la partie 5.4. Enfin, nous pourrons concevoir une architecture distribuée de processeurs d'AntX de sorte que chaque processeur traite une partie

de l'image. Ainsi, il sera possible de traiter des images de grandes tailles en un temps acceptable.

Durant ces cinq mois, j'ai pu acquérir une méthodologie sur la conception d'un état de l'art grâce à l'étude bibliographique menée en début du stage. De plus, l'utilisation de divers Makefile et scripts m'a permis de me familiariser avec de nouvelles méthodes de travail et de nouveaux langages de programmation. J'ai aussi pu mettre en application la formation acquise lors de mes études au sein de l'INSAT et de l'université paris sud 11. De plus, j'ai pu approfondir mes connaissances dans le domaine des processus de vision par ordinateur et la conception d'architecture numérique.

# Annexes

## Annexe A : Le tri à Bulles

Le tri à bulles ou tri par propagation est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau, comme les bulles d'air remontent à la surface d'un liquide.

Sa complexité est de l'ordre de  $n^2$  dans le pire des cas (où  $n$  est la taille du tableau), ce qui le classe parmi les mauvais algorithmes de tri.

Voici la description en pseudo-code du tri à bulle, pour trier un tableau  $T$  de  $n$  éléments numérotés de 0 à  $n-1$  :

```
procédure tri_bulle(tableau T, entier n)
    répéter
        aucun_échange = vrai
        pour j de 0 à n - 2
            si T[j] > T[j + 1], alors
                échanger T[j] et T[j + 1]
                aucun_échange = faux
        tant que aucun_échange = faux
```

## Annexe B : calcul des dérivées secondes et des dérivées premières

$$\frac{\partial^2 D}{\partial X^2} = \begin{pmatrix} \frac{\partial^2 D(x, y, \sigma)}{\partial x^2} & \frac{\partial^2 D(x, y, \sigma)}{\partial x \partial y} & \frac{\partial^2 D(x, y, \sigma)}{\partial x \partial \sigma} \\ \frac{\partial^2 D(x, y, \sigma)}{\partial x \partial y} & \frac{\partial^2 D(x, y, \sigma)}{\partial y^2} & \frac{\partial^2 D(x, y, \sigma)}{\partial y \partial \sigma} \\ \frac{\partial^2 D(x, y, \sigma)}{\partial x \partial \sigma} & \frac{\partial^2 D(x, y, \sigma)}{\partial y \partial \sigma} & \frac{\partial^2 D(x, y, \sigma)}{\partial \sigma^2} \end{pmatrix}$$

$$\frac{\partial D}{\partial X} = \begin{pmatrix} \frac{\partial D(x, y, \sigma)}{\partial x} \\ \frac{\partial D(x, y, \sigma)}{\partial y} \\ \frac{\partial D(x, y, \sigma)}{\partial \sigma} \end{pmatrix}$$

Et :

$$\frac{\partial^2 D(x, y, \sigma)}{\partial x^2} = 0.25 * [D(x + 2, y, \sigma) - 2D(x, y, \sigma) + D(x - 2, y, \sigma)]$$

$$\frac{\partial^2 D(x, y, \sigma)}{\partial y^2} = 0.25 * [D(x, y + 2, \sigma) - 2D(x, y, \sigma) + D(x, y - 2, \sigma)]$$

$$\frac{\partial^2 D(x, y, \sigma)}{\partial \sigma^2} = 0.25 * [D(x, y, \sigma + 1) - 2D(x, y, \sigma) + D(x, y, \sigma - 2)]$$

$$\frac{\partial^2 D(x, y, \sigma)}{\partial x \partial y} = 0.25 * [D(x + 1, y, \sigma) - D(x + 1, y - 1, \sigma) - D(x - 1, y + 1, \sigma) + D(x - 1, y - 1, \sigma)]$$

$$\frac{\partial^2 D(x, y, \sigma)}{\partial x \partial \sigma} = 0.25 * [D(x + 1, y, \sigma) - D(x + 1, y, \sigma - 1) - D(x - 1, y, \sigma + 1) + D(x - 1, y, \sigma - 1)]$$

$$\frac{\partial^2 D(x, y, \sigma)}{\partial y \partial \sigma} = 0.25 * [D(x, y + 1, \sigma) - D(x, y + 1, \sigma - 1) - D(x, y - 1, \sigma + 1) + D(x, y - 1, \sigma - 1)]$$

$$\frac{\partial D(x, y, \sigma)}{\partial x} = 0.5 * [D(x + 1, y, \sigma) - D(x - 1, y, \sigma)]$$

$$\frac{\partial D(x, y, \sigma)}{\partial y} = 0.5 * [D(x, y + 1, \sigma) - D(x, y - 1, \sigma)]$$

$$\frac{\partial D(x, y, \sigma)}{\partial \sigma} = 0.5 * [D(x, y, \sigma + 1) - D(x, y, \sigma - 1)]$$

# Annexe C : Prototypes des fonctions principales

Image intégrale :

```
void ImageIntegrale
(
    int width,
    int height,
    int **mosaic,
    int **image_integrale
);
```

Entrées :

- width et height : représentent respectivement la largeur et la longueur de l'image.
- mosaic : représente l'image source.

Sorties :

- image\_integrale : représente l'image intégrale de mosaic.

Fastessian pour un nombre d'octaves égal à 3 :

```
void Fasthessian_1
(
    int width,
    int height,
    int **image_integrale,
    MY_TYPE **det_00,
    MY_TYPE **det_01,
    MY_TYPE **det_02,
    MY_TYPE **det_03,
    MY_TYPE **det_04,
    MY_TYPE **det_05,
    MY_TYPE **det_06,
    MY_TYPE **det_07
);
```

Entrées :

- width, height et image\_integrale.

Sorties :

- det\_00, det\_01, det\_02, det\_03, det\_04, det\_05, det\_06 et det\_07 : ce sont les matrices contenant l'ensemble des points candidats. Ils sont le résultat de convolution des boites filtres avec l'image intégrale.

### cal\_det\_i

```
void cal_det_i
(
    int w,
    int h,
    int step,
    int filtre,
    MY_TYPE **det_i,
    int **image_integrale,
    int pas_filtre
);
```

Entrées :

- w et h : représentent respectivement la largeur et la longueur de l'image.
- image\_integrale
- step : représente l'échelle
- filtre : représente la taille du « boite filtre » à appliquer.
- pas\_filtre : (voir figure 46).

Sorties :

- det\_i : déterminant de la matrice hessienne.

### Filtre\_xx, Filtre\_yy et Filtre\_xy

```
int filtre_xx
(
    int
    **image_integrale,
        int filtre,
        int row,
        int col,
        int rows
);
```

```
int filtre_yy
(
    int
    **image_integrale,
        int filtre,
        int row,
        int col,
        int rows
);
```

```
int filtre_xy
(
    int
    **image_integrale,
        int filtre,
        int row,
        int col,
        int rows
);
```

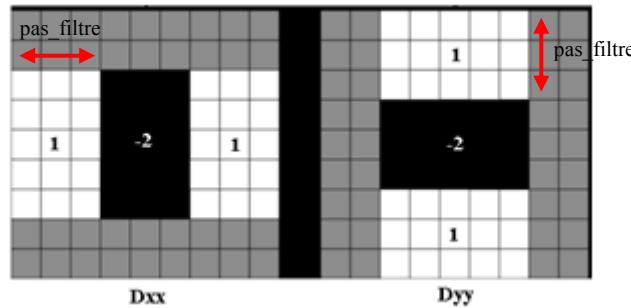


Figure 46 : « Boites filtres » de taille 9x9 selon la direction horizontale et verticale

## Lmls

```
void lmls_1
(
    int width,
    int height,
    MY_TYPE **determinant_0,
    MY_TYPE **determinant_1,
    MY_TYPE **determinant_2,
    MY_TYPE **determinant_3,
    MY_TYPE **Keypoint_1,
    MY_TYPE **Keypoint_2,
    int *nb_ligne1,
    int *nb_ligne2,
    int step
);
```

Entrées :

- Width, height, determinant\_0, determinant\_1, determinant\_2, determinant\_3 et step.

Sorties :

- Keypoint\_1 : contient l'ensemble des points d'intérêt détecté par comparaison des 3 premières matrices (voir figure « 24 »).
- Keypoint\_2 : contient l'ensemble des points d'intérêt détecté par comparaison des 3 dernières matrices (voir figure « 24 »).
- nb\_ligne1 : le nombre de points d'intérêt contenu dans Keypoint\_1.
- nb\_ligne2 : le nombre de points d'intérêt contenu dans Keypoint\_2.

### **max\_local:**

```
MY_TYPE max_local
(
    int j,
    int i,
    MY_TYPE **determinant_i,
    MY_TYPE max_i1,
    MY_TYPE max_i2,
    int step
);
```

Entrées :

- j, i : coordonnée du point central de la matrice 1.
- determinant\_i : matrice 1
- max\_i1, max\_i2 : valeur maximale de la matrice 0 et valeur maximale de la matrice 1.
- step

En sortie cette fonction renvoie la valeur du point s'il est extremum sinon elle renvoie 0.

### **Tri pour 3 octaves**

```
void tri
(
    MY_TYPE **Keypoint_1,
    MY_TYPE **Keypoint_2,
    MY_TYPE **Keypoint_21,
    MY_TYPE **Keypoint_22,
    MY_TYPE **Keypoint_31,
    MY_TYPE **Keypoint_32,
    MY_TYPE **Keypoint,
    int *nb_ligne1,
    int *nb_ligne2,
    int *nb_ligne21,
    int *nb_ligne22,
    int *nb_ligne31,
    int *nb_ligne32,
    int *nb_ligne
);
```

Entrées :

- Keypoint\_1, Keypoint\_2 : Contiennent les points d'intérêt détectés au niveau de l'octave1.

- Keypoint\_21, Keypoint\_22 : Contiennent les points d'intérêt détectés au niveau de l'octave2.
- Keypoint\_31, Keypoint\_32 : Contiennent les points d'intérêt détectés au niveau de l'octave3.
- nb\_ligne1 : représente le nombre de points d'intérêt contenu dans Keypoint\_1.
- nb\_ligne2 : représente le nombre de points d'intérêt contenu dans Keypoint\_2.
- nb\_ligne21 : représente le nombre de points d'intérêt contenu dans Keypoint\_21.
- nb\_ligne22 : représente le nombre de points d'intérêt contenu dans Keypoint\_22.
- nb\_ligne31 : représente le nombre de points d'intérêt contenu dans Keypoint\_31.
- nb\_ligne32 : représente le nombre de points d'intérêt contenu dans Keypoint\_32.

Sorties :

- Keypoint : contient l'ensemble des points d'intérêt à décrire
- nb\_ligne : représente le nombre de points d'intérêt contenu dans Keypoint.

## Descripteur

```
void descripteur
(
    int width,
    int height,
    MY_TYPE **Keypoint,
    int *nb_ligne,
    int **img
);
```

Entrées :

- width, height, img (image\_integrale), nb\_ligne et Keypoint.

Ci-dessous les prototypes des fonctions **reponse\_haar1XX**, **reponse\_haar1YY** et **Box\_integral**.

```
int reponse_haar1XX
(
    int row,
    int col,
    int **tab_tmp,
    int s
);
```

```
int reponse_haar1YY
(
    int row,
    int col,
    int **tab_tmp,
    int s
);
```

```
int Box_Integral
(
    int **img,
    int row,
    int col,
    int rows,
    int cols
);
```

row et col représente les coordonnées du point.

tab\_tmp : représente la fenêtre de taille  $20\sigma$  autour du point d'intérêt.

s : représente l'échelle.

rows et cols représentent la longueur et la largeur du filtre de Haar

## Annexe D : Images

### Motos



## Arbre



# Références bibliographiques

- [1] Richard P. Kleihorst, Anteneh A. Abbo, Vishal Choudhary, and Harry Broers, "Scalable IC Platform for Smart Cameras," *EURASIP Journal on Applied Signal Processing*, vol. 2005, no. 13, pp. 2018-2025, 2005.
- [2] H. P. Moravec. Rover visual obstacle avoidance. *In Proc. of the 7th International Joint Conference on Artificial Intelligence*, pages 785-790, 1981.
- [3] C. Harris, M. Stephens, "A combined corner and edge detector", *proceeding of the 4<sup>th</sup> Alvey Vision Conference*, pp.147-151, 1988.
- [4] Smith, S: A new class of corner finder. *In: British Machine Vision Conference* (1992).
- [5] T. Lindeberg, Feature Detection with Automatic Scale Selection, *International Journal of Computer Vision*, ISSN: 0920-5691, Volume 30, Issue 2, pp. 79-116, 1998.
- [6] K. Mikolajczyk et C. Schmid. An affine invariant interest point detector. *In Proceedings of European Conference on Computer Vision*, pages 128-142, 2002.
- [7] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal on Computer Vision*, 20 :91-110, 2003.
- [8] K.Mikolajczyk, C.Schmid, Scale & Affine Invariant Interest Point Detectors, *International Journal of Computer Vision*, ISSN:0920-5691, Volume 60 , Issue 1, pp. 63 - 86, 2004.
- [9] C. Schmid, R. Mohr, et C. Bauckhage. Comparing and evaluating interest points. *In Proc. of International Conference on Computer Vision*, 1998.
- [10] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir et L. Van Gool, A comparison of affine region detectors, *International Journal of Computer Vision*, ISSN: 0920-5691, volume 65(1/2), pp. 43-72, 2005.
- [11] J. Matas, O. Chum, M. Urban, et T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. *In Proceedings of the 13th British Machine Vision Conference*, Cardiff, UK, pages 384.393, 2002.
- [12] E. Rosten et T. Drummond. Machine learning for high-speed cornerdetection. *In Proceedings of European Conference on Computer Vision*, pages 430-443, 2006.
- [13] H. Bay, T. Tuytelaars, et L. Van Gool. Surf: Speeded up robust features. *European Conference on Computer Vision*, 1:404-417, 2006.
- [14] Krystian Mikolajczyk et Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.
- [15] Y. Ke et R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. *In CVPR (2)*, pp 506-513, 2004.
- [16] W. Freeman et E. Adelson. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891.906, 1991.
- [17] J. Koenderink et A. van Doorn. Representation of local geometry in the visual system. *Biological Cybernetics*, 55:367.375, 1987.
- [18] L. Van Gool, T. Moons, et D. Ungureanu. Affine / photometric invariants for planar intensity patterns. *In Proceedings of the 4th European Conference on Computer Vision*, Cambridge, UK, pp 642-651, 1996.

- [19] E. Tola, V. Lepetit et P. Fua. A Fast Local Descriptor for Dense Matching. *In proceedings of Computer Vision and Pattern Recognition*, 2008.
- [20] A. Gil, O. Martinez Mozos, M. Ballesta, et O. Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual SLAM. *Machine Vision and Applications*, pp 1432-1769, 2009.
- [21] G. J. Burghouts et J.-M. Geusebroek, “Performance Evaluation of Local Invariants,” *Computer Vision and Image Understanding*, vol. 113, pp. 48-62, 2009
- [22] S. Belongie, J. Malik, et J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4), pp 509-522, 2002.
- [23] Paul A. Viola, Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. *In proceedings of CVPR (1)*, pp 511-518, 2001.
- [24] T. Tuytelaars, K. Mikolajczyk. Local Invariant Feature Detectors: A survey. *Foundation and Trends in Computer Graphics and Vision*, vol. 3, pp 177-280, 2008.
- [25] R. David, S. Chevobbe, Y. Lhuillier, Ph. Fauvel, et F. Pasquet. Antx, Architecture and Compilation Overview. *Technical report*, CEA LIST, 2009.