Architecture des traitements matériels (IP)

4.1 Vue d'ensemble

L'organisation de notre architecture, déployée sur FPGA, est représentée en figure 4.1. Elle se compose de plusieurs IP connectées entre elles, reliées à la caméra et au système logiciel par des interfaces dédiées.

Cette partie matérielle génère, à partir de l'image de la caméra, plusieurs résultats qui sont lus par le système logiciel embarqué :

- six Différences de Gaussiennes représentant six octaves de fréquence spatiale
- les listes triées de points d'intérêt trouvés dans ces six DoG

Un signal indique au système logiciel que les résultats sont disponibles, afin qu'ils les lise. Ainsi, il lit les coordonnées des points d'intérêt, et récupère dans les DoG les anneaux de pixels dont il a besoin pour générer les caractéristiques locales. Ces imagettes log-polaires sont alors envoyées au réseau de neurone artificiel par le biais du module Ethernet de la carte.

Le flux de pixels en provenance de l'interface de la caméra est transféré en entrée de la première IP : l'intensité de gradient. Un signal Enable permet d'activer le fonctionnement de cette IP dès qu'un nouveau pixel est disponible en entrée à chaque nouveau cycle d'horloge. Les coordonnées des pixels permettent aux IP de maîtriser les effets de

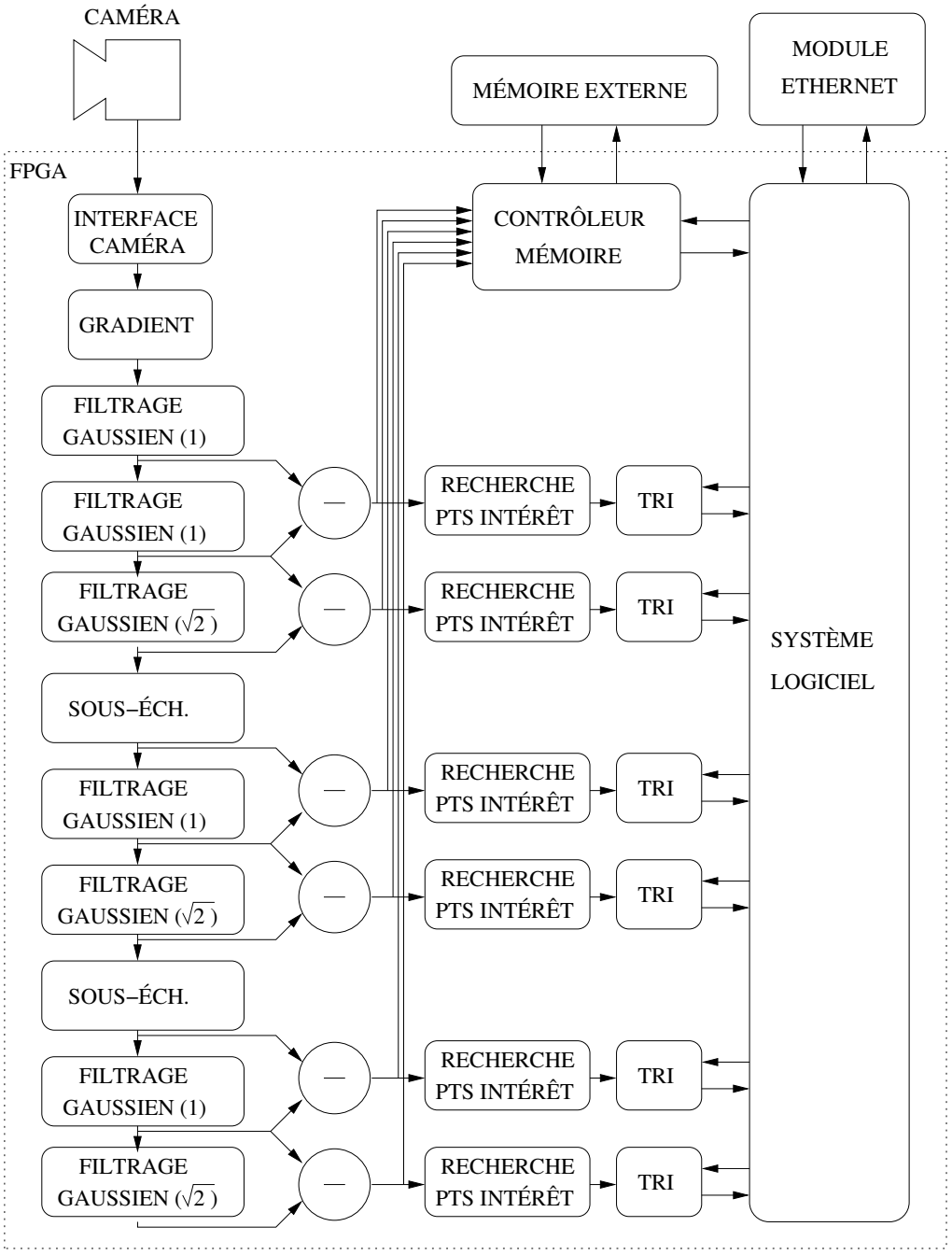

Figure 4.1 – **Vue d'ensemble des IP** - Organisation des unités de traitement matérielles dans le FPGA

bord. L'interface entre la caméra et l'IP de gradient génère ces signaux supplémentaires en fonction des réglages de la caméra et de ses signaux de sortie.

Les DoG sont stockées dans une mémoire externe au FPGA, un contrôleur mémoire dédié permet de stocker les six DoG, en laissant un accès en lecture pour le système logiciel.

Les pixels, leurs coordonnées et les signaux Enable forment l'ensemble des données communiquées d'une IP à l'autre, du gradient jusqu'aux sorties des DoG. Les recherches et tris de points d'intérêt ont des fonctionnements différents : l'IP de recherche fournira, en plus des pixels et de leurs coordonnées, un signal indiquant si le pixel est un point d'intérêt. Les IP de tri ne liront les pixels et leurs coordonnées que quand ce signal sera actif, afin de ne trier que les points d'intérêt.

La partie logicielle verra les IP de tri comme autant de mémoires RAM, contenant pour chaque octave les coordonnées des points d'intérêt.

4.2 Squelette des IP

La plupart des IP ayant pour entrées et sorties des flux de pixels de même type, il est utile de mettre en place une interface standard commune à ces IP. Cette standardisation permet une connexion transparente d'une IP à une autre, et la modularité de l'architecture globale du système matériel.

Cette interface se base sur les signaux suivants :

- la valeur de pixel,
- ses coordonnées X et Y,
- un signal Enable

Ainsi, chacune de ces IP traite ainsi un flux de pixels dont les coordonnées sont connues à tout moment, et un signal Enable permet de limiter le fonctionnement de l'IP que lorsqu'un pixel valide est fourni.

Les IP générant ce même type de données en sortie gèrent en interne leur latence intrinsèque, pour assurer la synchronisation des valeurs et des coordonnées des pixels de sortie. Un signal enable est généré afin de permettre à l'IP cliente de récupérer les

pixels seulement quand ils sont valides. L'interruption du flux de pixels en entrée sera alors répercuté de la première IP aux suivantes par effet boule de neige.

Le squelette externe d'une IP ayant cette interface standard en entrée et en sortie est présenté en figure 4.2.

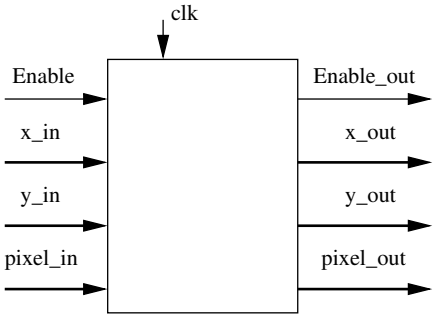

Figure 4.2 – **Interface des IP en flot de pixels** - Interface standard du système de vision

Les différences de Gaussiennes sont un cas particulier, les entrées de l'IP étant deux flux de pixels non-synchronisés. La section 4.5 montre comment la synchronisation des deux flux est gérée.

L'IP de recherche de points d'intérêt utilise cette interface en entrée, mais en sortie, le signal Enable_out est remplacé par un signal "Trier". Ce signal indique que le pixel en sortie est un point d'intérêt potentiel, et doit être inséré dans la liste triée de points d'intérêt (si sa valeur lui permet de rester dans la liste).

Les IP de tri de points d'intérêt fournissent un seul signal de façon active : une fois que l'image est traitée dans son intégralité, le signal "Terminé" indique à l'interface du système logiciel que la demi-échelle correspondante est traitée. Le système logiciel peut alors travailler sur chacune des six demi-échelles au fur et à mesure qu'elles finissent d'être étudiées par les IP. Les résultats des IP de tris, sont lus à la manière de mémoires RAM, contenant les coordonnées des points d'intérêt par ordre de valeurs de points d'intérêt.

4.3 Intensité de gradient

L'algorithme choisi pour calculer l'intensité de gradient de l'image d'entrée correspond à une version simplifiée de l'opérateur de Sobel. La différence fondamentale avec l'opérateur de Sobel est la racine carrée, que l'on remplace par une simple moyenne pour gagner en place sur le FPGA pour un moindre impact sur l'efficacité du système. Dans le cas où l'on voudrait éviter cette approximation, le déploiement d'un opérateur de racine carrée pourrait être envisagé. Différentes IP présentes dans la littérature permettent le calcul d'une racine carrée en une seul itération (CL00), ce qui permettrait de conserver une homogénéité du comportement temporel des IP du système de vision. En figure 4.3, on peut voir l'architecture proposée pour l'IP d'intensité de gradient.

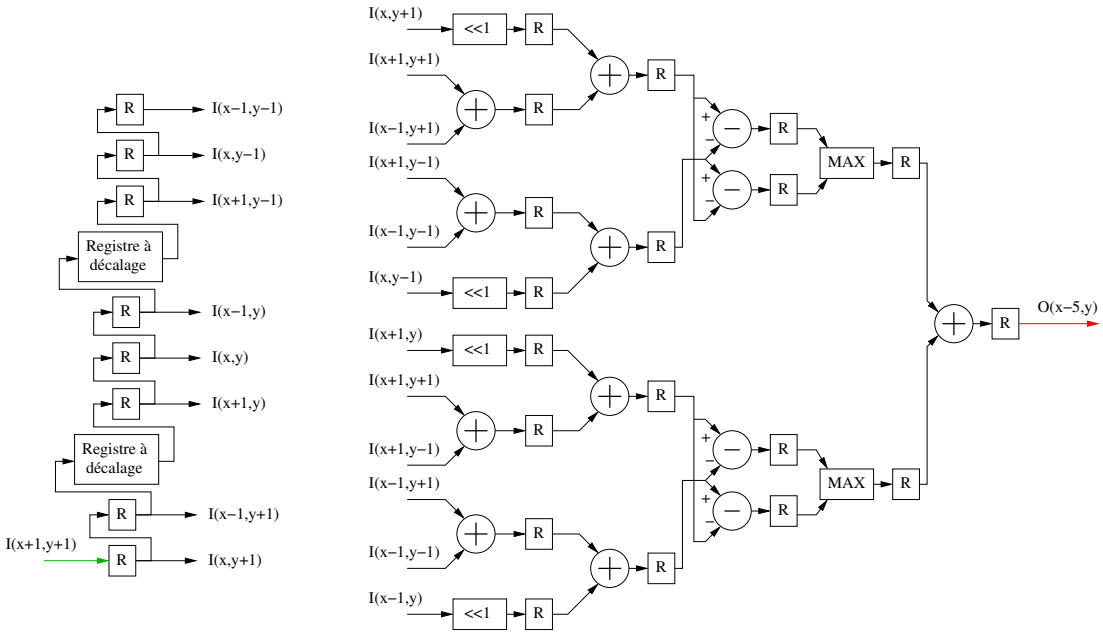

FIGURE 4.3 – **IP d'intensité de gradient** - À gauche, gestion des pixels d'entrée, à droite, calcul de l'intensité du gradient

La synchronisation des pixels en entrée de l'opérateur de l'IP est un point important. Une mémorisation des pixels d'entrée de l'IP, à base de registres à décalage et de simples registres, permet de fournir simultanément les huit pixels nécessaires au calcul de chaque pixel de sortie.

Des étages de registres permettent de disposer le calcul en pipeline, afin d'augmenter la bande passante, au prix d'un peu de latence supplémentaire. La latence de l'IP est ici de $w_{image} + 6$ cycles d'horloge, la mise en mémoire de deux lignes et un pixel étant nécessaire pour effectuer le calcul, et les étages de pipeline du calcul induisant un retard de seulement 5 cycles d'horloge.

La gestion des effets de bords dans ce bloc fonctionnel permet d'éviter la corruption d'une bande d'un pixel de large autour de l'image. Le mécanisme utilisé ici n'est pas montré dans la figure, faute de place. Il consiste tout simplement à remplacer par la valeur "0" tout pixel d'entrée du bloc de calcul qui serait hors de l'image. La détection des bords est effectuée à partir des coordonnées du pixel d'entrée, et des constantes w_{image} et h_{image}. Ce mécanisme de gestion d'effets de bords a pour conséquence d'atténuer la valeur du gradient pour les pixels disposés sur le bord de l'image, sans affecter les autres pixels, qui ne subissent pas d'effet de bord.

4.4 Filtrage Gaussien

Le filtrage Gaussien est réalisé par une convolution de l'image d'entrée avec un noyau de coefficients 2D. Étant donné que le filtrage Gaussien 2D est séparable en deux convolutions 1D, des déploiements en deux passes 1D sont étudiés et comparés aux équivalents en une passe 2D. On peut ainsi connaître l'impact du choix d'un filtrage séparable, et le surcoût qu'impliquerait un changement pour un filtrage non-séparable.

Les IP de convolution 2D présentes dans la littérature sont présentées en section 4.4.1. Il est à noter que la plupart de ces déploiements ne semblent pas particulièrement s'attacher à une optimisation quelconque de l'IP.

Les algorithmes décrits en section 3.1.2.1 sont tous fidèles à la version logicielle du filtrage Gaussien, au détail près du type de données (virgule fixe). Les différentes architectures proposées ici jouent sur l'organisation des opérations de ces algorithmes, afin de paralléliser les calculs au maximum.

Les types de données utilisés ici sont génériques, et aisément réglables avant synthèse des IP. Dans les tests qui ont été faits, les pixels d'entrée et de sortie sont stockés sur

des nombres à virgule fixe non-signés de 16 bits. Les produits et les sommes sont quant à eux stockés sur 32 bits.

4.4.1 État de l'art : convolution Gaussienne 2D

Si l'intensité de gradient, les DoG et sous-échantillonnages sont relativement simples à déployer, ou suffisamment peu communs sous forme d'IP, la convolution 2D représente un point plus délicat. Dans le cas présent, on souhaite :

- une bonne gestion des effets de bords, afin de minimiser l'erreur dans les résultats obtenus.
- une cadence d'exécution à la hauteur du reste du système : dans le cas présent, un nouveau pixel sera traité à chaque cycle d'horloge.
- une consommation raisonnable des ressources du FPGA ciblé.

Dans (BPS98), une architecture rapide est proposée pour déployer une convolution 2D sur un flot de pixels. Le calcul de la convolution se fait au fur et à mesure que les pixels entrent dans le système (calcul en pipeline), à une cadence d'un pixel par coup d'horloge. Cette architecture, proposée en 1998, est déployée sur les FPGA de cette époque, disposant de peu de mémoire. L'auteur propose de paralléliser la convolution sur plusieurs FPGA pour pallier à ce problème. Encore aujourd'hui, la parallélisation sur plusieurs FPGA reste envisageable dans le cas d'une fenêtre de convolution de très grande taille.

Dans (Nel00), une IP de convolution 2D 3×3 est présentée, accompagnée du code VHDL correspondant. L'approche traditionnelle est utilisée : les pixels en entrée passent par des registres et des FIFO pour que les entrées des multiplieurs correspondent aux pixels recouverts par la fenêtre de convolution. Les produits sont ensuite sommés dans un arbre d'additionneurs. La division par la somme des coefficients est approchée à la puissance de 2 la plus proche, l'erreur étant jugée acceptable par l'auteur. Son approche permet une gestion basique des effets de bords : dès que la fenêtre de convolution dépasse du bord de l'image, le résultat de convolution est rendu nul.

L'étude de la convolution 2D a aussi fait l'objet d'optimisations énergétiques (Per03). L'architecture proposée par Perri se base sur des multiplieurs SIMD, sur un arbre d'ad-

ditionneurs SIMD et un module de saturation SIMD (ce dernier servant à formater la somme des produits sur 16 bits non-signés). Cette architecture est adaptable à de nouveaux paramètres en cours d'exécution, que ce soit la largeur des bus de données des pixels ou des coefficients du noyau de convolution.

Une méthode de déploiement matériel de la convolution 2D, proposée par B. Cope (Cop06), apparaît comme très proche de ce que l'on souhaite obtenir. Cette architecture utilise la flexibilité des FPGA pour déployer des pipelines parallèles pour le calcul des convolutions. L'auteur compare cette architecture à des déploiements logiciels sur PC et sur GPU. Si les déploiements sur certains GPU se montrent bien plus efficaces que l'architecture FPGA proposée pour déployer des convolutions pour des petites fenêtres, l'avantage revient au FPGA dès que la fenêtre de convolution utilisée est de 4×4 pixels. Alors, un déploiement sur Spartan3 prend de vitesse le GPU 6800 Ultra qui travaillait 5.6 fois plus vite que lui pour une fenêtre de 2×2, et 2.1 fois plus vite pour une fenêtre de 3×3. Les tailles de fenêtres concernant notre projet étant de 7×7 et 9×9, le facteur d'accélération correspondant est respectivement de 4.1 et de 8.1 en faveur du déploiement FPGA.

Plus récemment, Fons et Al. (FFC10) présentaient une architecture de convolution 2D reconfigurable dynamiquement. Les dimensions du noyau, les largeurs de bus et les étages de pipeline sont ici reconfigurables en cours d'exécution, en exploitant la technologie de reconfiguration dynamique des Virtex 4 de Xilinx. Les auteurs présentent les FPGA comme étant toujours plus efficaces que les DSP récents pour la convolution 2D, ce qui ne peut que renforcer les conclusions de Cope. Une fois de plus, la somme des produits est réalisée à l'aide d'un arbre d'additionneurs.

La capacité de fournir un pixel de l'image de sortie à chaque coup d'horloge est proposée dans chacun de ces documents. Cet aspect est attrayant pour une plate-forme dont la résolution d'images d'entrée n'est pas fixée définitivement. En effet, si une image de faible résolution ne représente que peu de pixels à traiter, on préférera garder la possibilité de traiter des images de grandes dimensions tout en gardant la même cadence de fonctionnement. La limitation sera ici la fréquence de fonctionnement maximale de l'architecture sur le FPGA. À l'inverse, l'utilisation d'une caméra de faible résolution

permettra le déploiement à plus faible cadence, ce qui permet généralement de réduire la puissance électrique consommée.

Dans le système robotique (BMC08) présenté en 2.1.1, les auteurs présentent un déploiement matériel de détection de caractéristiques à base de DoG. Leur déploiement des filtrages Gaussiens ne s'effectue pas en 2D directement, mais en deux passes 1D. L'utilisation de deux passes 1D implique le stockage d'au moins une partie du résultat intermédiaire dans une mémoire, ce qui peut se révéler délicat à mettre en œuvre dans un état d'esprit d'économie de mémoire. Les résultats de ce déploiement sont présentés, l'opérateur étant capable de traiter un filtrage Gaussien de 30 images de 320×240 pixels par seconde.

4.4.2 Noyau de convolution et division

Les noyaux des convolutions Gaussiennes utilisés dans l'application sont définis sous forme de tableaux de constantes dans le code VHDL. Les coefficients sont déclarés sur 16 bits chacun, étant donné que les pixels sont stockés sur 16 bits et que la somme des produits sera sur 32 bits. Afin d'économiser des ressources utilisées par les multiplications, l'outil de synthèse réduira automatiquement la taille des bus de ces constantes à des valeurs adéquates (un coefficient de valeur 11 n'a besoin d'être stocké que sur 4 bits par exemple).

Pour toutes les architectures proposées, un élément reste parfaitement identique : la division finale de la somme des produits. Une division est une opération sensible quand il s'agit de la déployer sur une architecture électronique. Au contraire, une division par une puissance de 2 est une opération triviale : à partir d'une valeur, un décalage à droite de n permet d'obtenir la division par 2^n. Si l'on suppose que le diviseur est constant, l'exclusion des n bits de poids faible de la valeur d'entrée permet de faire cette division sans la moindre latence. Dans le cas contraire, il est nécessaire d'utiliser un opérateur de division coûteux en ressources matérielles. Un remplissage aura lieu si besoin par câblage direct (dans le cas présent, on garde les 16 bits de poids fort d'une valeur de 32 bits, aucun remplissage n'est nécessaire).

Les coefficients du noyau Gaussien seront calculés de façon à ce que leur somme soit égale à une puissance de 2 (ou le plus proche possible). On peut ainsi normaliser la sortie de convolution par une simple division par cette puissance de 2, ce qui réduit l'utilisation des ressources du circuit. Nelson, par exemple, se base sur ce principe (Nel00), sans adapter les valeurs des coefficients : si la somme des coefficients vaut 9, il propose de diviser par 8. Dans le cas présenté, cette approche nécessite un bit supplémentaire en sortie pour gérer les éventuels débordements, ou réduit la précision du résultat de moitié si l'apparition de ce bit supplémentaire ampute le résultat de son bit de poids faible. L'obtention de valeurs maximales du gradient (tous les bits à '1') étant mathématiquement impossible avec l'opérateur de Sobel (maximum théorique composé de 3 bits de poids faible à '0'), et une image maximisant le résultat de l'opérateur étant quasi-impossible à capter sur une caméra hors conditions de laboratoire (optiques et capteurs imparfaits, bruit électronique), une marge d'erreur est acceptable dans les valeurs de coefficients utilisés. On tendra alors vers la valeur la plus juste des coefficients, pour se rapprocher au mieux des résultats obtenus en logiciel avec des coefficients de type "float" ou "double".

Dans le cas présent, on suppose la somme des coefficients stockée sur 24 bits et les pixels sur 17 bits, la somme des coefficients doit se rapprocher de 2^{24}. Le tableau 4.1 répertorie deux types d'erreurs obtenues pour différentes largeurs de noyaux de convolution. La première, l'erreur par rapport à 2^{24}, indique quel est le rapport entre cette valeur idéale pour la somme des coefficients (1000000 en base 16). Cette erreur est due à la baisse de précision que l'on subit en passant d'un "double" à un entier proche de 2^{24}. La deuxième erreur est mesurée par rapport à la valeur de la somme des coefficients pour un noyau de dimensions infinies (logiciellement approché avec des noyaux de 2001×2001 coefficients). Cette dernière erreur indique donc l'erreur de précision due à la taille du noyau, combinée à la première erreur.

Au delà d'une certaine taille de noyau, les coefficients sont tous nuls, dépasser cette taille limite n'apporte rien à la précision du résultat. Pour $\sigma = 1$, un noyau de 9×9 suffit à contenir tous les coefficients pour une somme proche de 2^{24}. Pour $\sigma = \sqrt{2}$, cette taille limite est de 15×15 coefficients. Les précisions maximales de $-6.15e^{-8}$ pour $\sigma = 1$ et de $-7.84e^{-9}$ pour $\sigma = \sqrt{2}$ sont listées ici pour des coefficients codés sur 24 bits au plus. Pour aller au delà de cette précision, il faut utiliser des bus de données plus larges

K	σ^2	somme des coefficients	erreur par rapport à 2^{24}	erreur par rapport à la Gaussienne réelle
3	1	FFFFFF	$-5.96e^{-10}$	$-2.21e^{-3}$
5	1	FFFFFA	$-3.58e^{-9}$	$-1.82e^{-4}$
7	1	FFFFFF	$-5.96e^{-10}$	$-5.41e^{-6}$
9	1	FFFFFD	$-1.79e^{-9}$	$-6.15e^{-8}$
3	2	1000000	0	$-4.79e^{-3}$
5	2	FFFFFE	$-1.19e^{-9}$	$-1.37e^{-3}$
7	2	FFFFFC	$-2.38e^{-9}$	$-2.29e^{-4}$
9	2	FFFFFF	$-5.96e^{-10}$	$-2.32e^{-5}$
11	2	100000E	$8.34e^{-9}$	$-1.44e^{-6}$
13	2	FFFFF4	$-7.15e^{-9}$	$-6.24e^{-8}$
15	2	FFFFF5	$-6.56e^{-9}$	$-7.84e^{-9}$

TABLE 4.1 – **Précision des noyaux Gaussiens** - noyaux de convolution de K^2 coefficients sur 24 bits : erreur de la somme par rapport à 2^{24}, et erreur de précision de la convolution

pour stocker les coefficients, l'agrandissement des noyaux n'ayant aucun effet sans cette mesure supplémentaire.

4.4.3 Convolution 2D

Plusieurs architectures matérielles ont été évaluées pour la convolution 2D.

La première solution est la méthode traditionnellement présente dans la littérature (section 4.4.1), dont l'architecture est présentée en figure 4.4.

Cet opérateur calcule à chaque coup d'horloge le pixel de sortie en fonction des pixels d'entrée qui lui correspondent, et de leurs coefficients respectifs. Les pixels de l'image d'entrée passent par un jeu de registres à décalage pour arriver en entrée des multiplieurs simultanément. Les multiplications se font ainsi en parallèle pour un même pixel de l'image de sortie. Les produits sont ensuite additionnés entre eux pour produire la valeur du pixel de sortie.

La latence L de cette IP peut se calculer, en cycles d'horloge, par la formule $L_{IP} = (H_{noyau}/2) * W_{image} + (W_{noyau}/2) + L_{opérateur}$ (divisions entières). Pour un noyau de convolution de 9×9 pixels, $L_{opérateur}$ sera de 8 coups d'horloge : 7 étages de

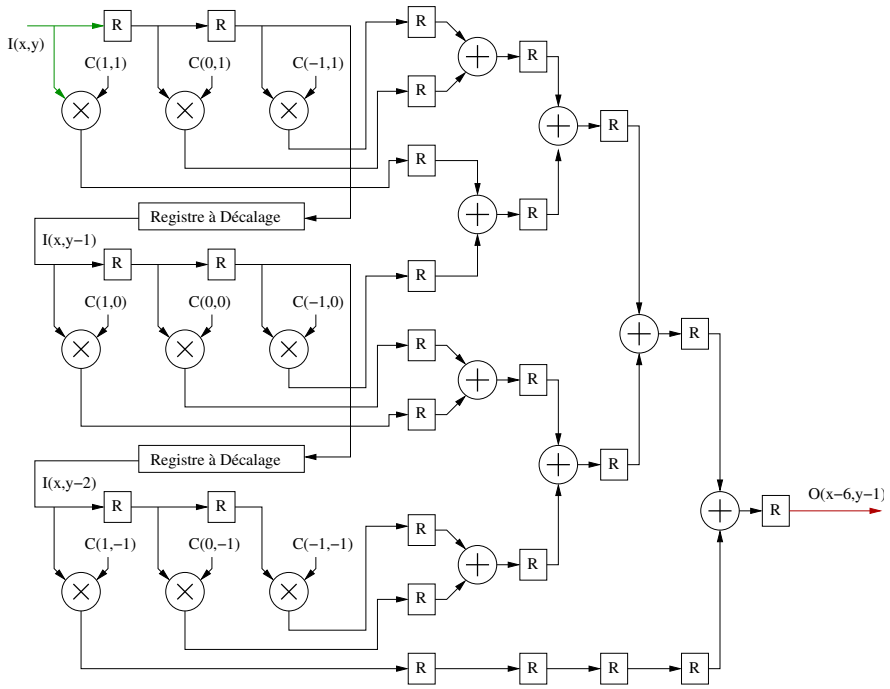

Figure 4.4 – **Convolution 2D matérielle : Architecture traditionnelle** - Exemple sur noyau 3x3

pipeline permettant de calculer la somme des 81 produits, et ces derniers représentent un étage supplémentaire de pipeline. Le nombre d'étages dans le pipeline d'addition est facile à trouver. Pour calculer la somme de $2^7 = 128$ valeurs deux à deux, il y aura 7 étages de pipeline. Avec 6 étages, on ne peut calculer la somme que de $2^6 = 64$ valeurs. Pour additionner nos 81 produits, on a donc 7 étages de pipeline (en plus de l'étage de sortie de multiplication). Pour un noyau de 7×7 coefficients, les 49 valeurs à additionner nécessitent 6 étages de pipeline. Pour une image 320x240 sur laquelle on utilise un noyau de coefficients de taille 9x9, la latence de l'IP sera de 4x320 + 4 + 8 =1292 coups d'horloge, en supposant que le flot de pixel envoie un nouveau pixel à chaque coup d'horloge. Sur ces 1292 coups d'horloge, 1284 seront dus aux dimensions du noyau de coefficients. Pour calculer le pixel de sortie (produit de convolution) aux coordonnées [x,y], tous les pixels de l'image d'entrée devront être lus par l'IP. D'où le décalage de 4 pixels en x et en y. Seuls 8 coups d'horloge seront dus à la topologie de l'opérateur.

La deuxième solution, utilise le principe des MACC pour cette convolution 2D. Deux architectures basées sur ce principe sont présentées en figure 4.5.

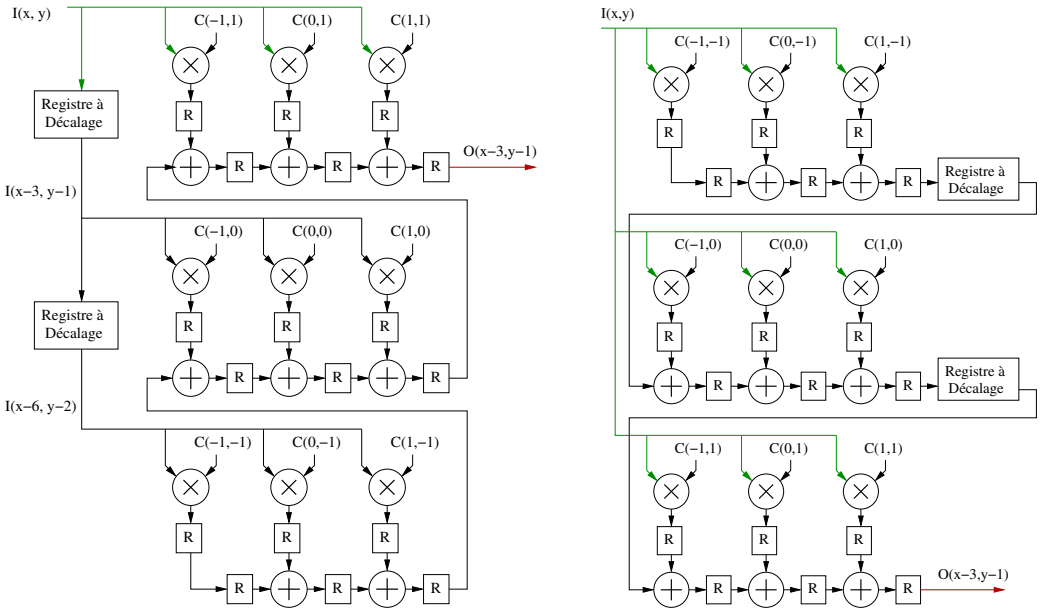

FIGURE 4.5 – **Convolution 2D matérielle : Architectures type MACC** - Exemples sur noyau 3x3. Mise en registre des pixels d'entrée ou de l'accumulation. Opérateurs MACC représentés par leurs multiplieurs et additionneurs internes

Cette fois, la somme des produits sera calculée selon le principe des MACC. La somme des produits est calculée par accumulations successives. Dans l'architecture de gauche, le dernier pixel sera multiplié simultanément par tous les coefficients de la dernière ligne du noyau. L'architecture MACC permet d'additionner les résultats obtenus aux sommes de produits correspondantes. Des registres à décalage permettent de multiplier avec les autres lignes de coefficients du noyau, les pixels adéquats. Cette architecture permet de stocker en registres à décalage les pixels d'entrée, qui ont souvent un bus moins large que les résultats de MACC.

L'architecture de droite fonctionne sur le même principe, à la différence que l'utilisation de registres à décalage se fera sur les résultats de MACC. Le pixel d'entrée de l'IP est multiplié par tous les coefficients du noyau, et les résultats sont accumulés de façon à obtenir à chaque coup d'horloge un pixel en sortie.

Le pixel de sortie de ces deux architectures correspond simplement à la sortie de la dernière MACC.

La latence L de ces IP se calculera par la formule $L_{IP} = H_{noyau} * W_{image} + W_{noyau} + L_{opérateur}$, comme pour l'autre solution. La différence réside dans la valeur de $L_{opérateur}$, qui voit sa valeur baisser à 2, quelle que soit la taille du noyau de convolution : un registre en sortie de multiplication, un autre en sortie d'accumulation.

Les trois solutions proposées utilisent le même nombre de multiplieurs et d'additionneurs, seule la topologie de l'opérateur diffère.

La première est plus intuitive, cependant les solutions à base de MACC sont plus pratiques à déployer sous forme d'IP générique : on souhaite pouvoir utiliser le même code pour différentes tailles de noyau et de bus de données. Les solutions à base de MACC permettent aussi de réduire légèrement la latence des calculs, et surtout de s'abstraire de la variation de latence introduite par l'arbre d'additionneurs. En effet, la somme des produits est calculée progressivement, les additionneurs étant disposés non plus en arbre mais en ligne. La taille du masque de convolution, qui impacte directement sur la latence de la première architecture, n'aura pas d'effet sur la latence des deux autres. À partir du moment où l'opérateur aura reçu le dernier pixel nécessaire au calcul d'une fenêtre, le pixel de sortie correspondant au centre de la fenêtre sera calculée en

deux coups d'horloge. Cette latence est due aux deux registres de la dernière MACC (sortie de multiplieur et sortie d'additionneur).

De plus, vis-à-vis du code VHDL, la généricité des dimensions du noyau Gaussien est rendue plus aisée, les additions étant intégrées à l'architecture en tableau 2D du reste des opérations. Pour la première solution, la conception générique d'un arbre d'additionneurs n'est pas un problème trivial, ces opérations formant une architecture irrégulière dès que le nombre d'additionneurs n'est pas une puissance de 2.

Pour les résultats des produits et des sommes de produits, on pourra utiliser des données plus larges que pour les pixels d'entrée et de sortie, afin d'améliorer la précision de la convolution. Cette décision entraînerait l'avantage des deux premières solutions proposées, car la mémoire consommée par la troisième augmenterait proportionnellement avec la largeur des données des MACC. Dans le cas où ces valeurs seraient stockées sur des données de même taille, les ressources mémoire nécessaires aux deux solutions seraient équivalentes, et la troisième solution garde un avantage conséquent, présenté dans la section suivante.

4.4.4 Coefficients redondants dans les noyaux de convolution

Dans beaucoup de cas nécessitant une convolution 2D, et particulièrement dans le cas présent, plusieurs coefficients ont la même valeur au sein d'un noyau de convolution. Dans le cas d'un noyau Gaussien, une très forte symétrie est visible : les coefficients sont répétés 8 fois, sauf sur les diagonales, la verticale, et l'horizontale qui se croisent au centre, dont les coefficients sont répétés 4 fois. Le centre, lui, est la seule valeur unique du noyau. Dans un tel cas, il est possible de réduire fortement le nombre de multiplieurs dans l'IP, en utilisant un même multiplieur pour calculer le produit d'un coefficient avec la somme des pixels concernés. Le multiplieur aura un bus de données plus large en entrée (une somme de pixels nécessitant un bus plus large qu'un pixel seul), c'est là le défaut de la mise en commun de multiplieurs. Cette solution est notamment proposée par French (Fre04) pour la convolution 2D.

On peut le remarquer en figure 4.6, où l'on passe de 9 à 3 multiplieurs en regroupant les coefficients de même valeur, et en multipliant la somme des pixels concernés par ces coefficients.

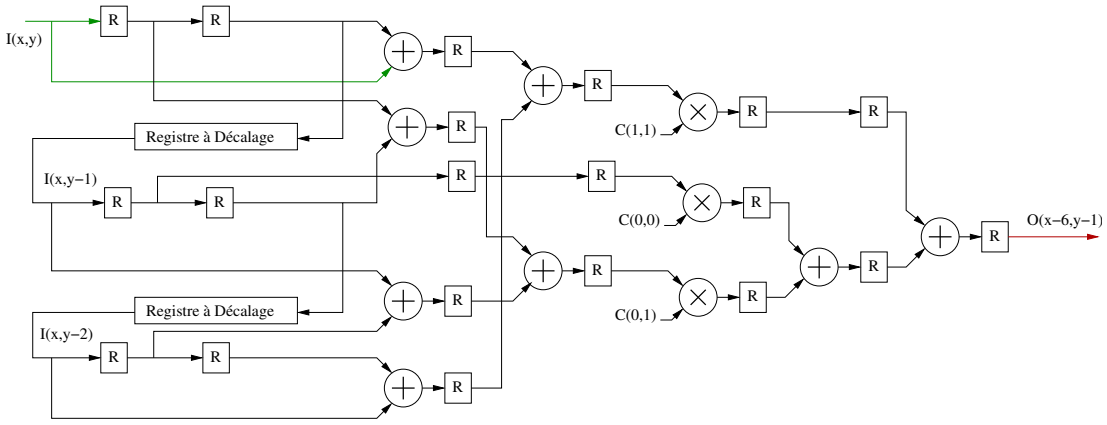

FIGURE 4.6 – **Convolution 2D matérielle : Coefficients redondants - architecture traditionnelle** - Exemple sur noyau Gaussien 3x3

Cette solution garde les défauts de l'architecture traditionnelle, à savoir la difficulté de conception générique, et la latence plus forte et dépendante de la taille du noyau.

Concernant les architectures à base de MACC, en figure 4.7, on peut voir que la solution de droite permet une réduction plus efficace du nombre de multiplieurs.

La solution de gauche permet de passer de 9 à 6 multiplieurs, les coefficients pouvant être regroupés par ligne, mais pas d'une ligne à l'autre dans cette architecture. La solution de droite permet la même réduction que l'architecture traditionnelle, au détriment de la mémoire, dans le cas où les bus de données des MACC seraient plus larges que ceux des pixels.

Un compromis entre les trois éléments (latence et généricité, utilisation mémoire, nombre de multiplieurs) doit donc être choisi.

Il reste une optimisation essentielle dans le cas de noyaux de convolution Gaussiens : la séparation en deux convolutions 1D.

4.4.5 Séparation en deux convolutions 1D

Le filtrage Gaussien par convolution 2D étant séparable en deux passes de convolution 1D, des IP correspondantes ont été développées, afin de les comparer aux convolu-

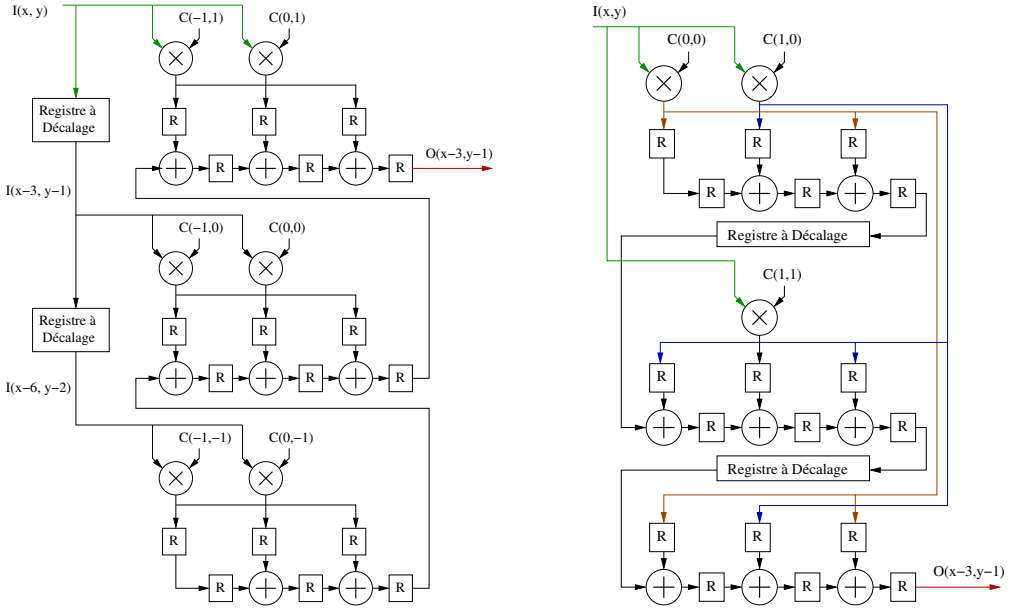

FIGURE 4.7 – **Convolution 2D matérielle : Coefficients redondants - architectures MACC** - Exemple sur noyau Gaussien 3x3

tions 2D. La figure 4.8 présente l'architecture de ces IP.

Dans le cas général, on remarque que le nombre d'opérateurs est diminué par rapport aux architectures 2D. Cependant, dans le cas du noyau de 3×3 coefficients, un plus grand nombre de multiplieurs est nécessaire (4 au lieu de 3 dans le cas de noyaux de coefficients symétriques). Au delà de noyaux de 3×3 coefficients, les IP utilisant la séparation en deux passes 1D sont avantageuses ou similaires en tout point. L'application de vision utilisant des noyaux de tailles 7×7 et 9×9, cette architecture est avantageuse en tous points pour le SoC de vision.

À part l'utilisation d'un plus grand nombre de multiplieurs pour les petits noyaux de convolution, cette IP demeure plus intéressante que les solutions basées sur une seule passe 2D, quelle que soit la taille du noyau. Le choix d'un filtrage en deux passes 1D ou en une passe 2D dépendra, pour les petits noyaux, des ressources disponibles dans le FPGA au moment de la synthèse de l'architecture complète.

4.4.6 Gestion des effets de bords

Des effets de bord sont générés par toutes les IP de convolution décrites précédemment. La figure 4.9 permet de visualiser la cause de ces effets de bords : les pixels en

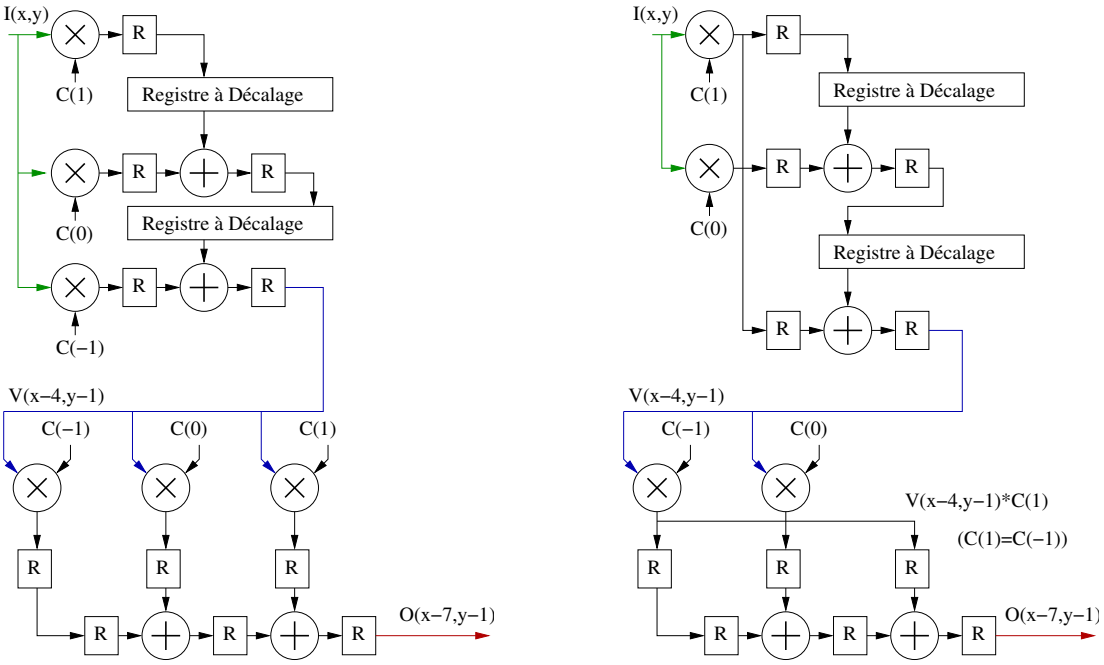

FIGURE 4.8 – **Convolution 2D matérielle : deux passes 1D** - Coefficients symétriques à droite. Le signal V correspond à la première passe, soit le filtrage vertical

noir sont hors de l'image. Lors du filtrage Gaussien, l'opérateur lit une fenêtre de $m \times n$ pixels dans l'image d'entrée (m et n étant les dimensions du noyau de coefficients). Si la plupart du temps, tous ces pixels seront valides, lorsque l'opérateur s'approche des bords de l'image, cette fenêtre dépasse plus ou moins de l'image d'entrée. En se rapprochant d'un bord de l'image, avec un noyau de 9×9 coefficients, on aura d'abord 9, puis 18, 27 et enfin 36 pixels invalides sur 81. Dans un coin de l'image, on arrivera dans les mêmes conditions à 56 pixels invalides sur 81. Les signaux des pixels pouvant être remplacés par des valeurs aléatoires (sur les bords haut et bas de l'image), ou des pixels de l'autre bord de l'image (sur les bords gauche et droit), la valeur de sortie de l'opérateur peut être fortement corrompue. Il est donc indispensable de prendre en compte ces effets de bord.

Deux solutions se présentent : une solution mathématiquement sans erreur, et une solution permettant de réduire la corruption des pixels de sortie proches du bord. La solution sans erreur nécessite de rogner la taille de l'image de sortie par rapport à

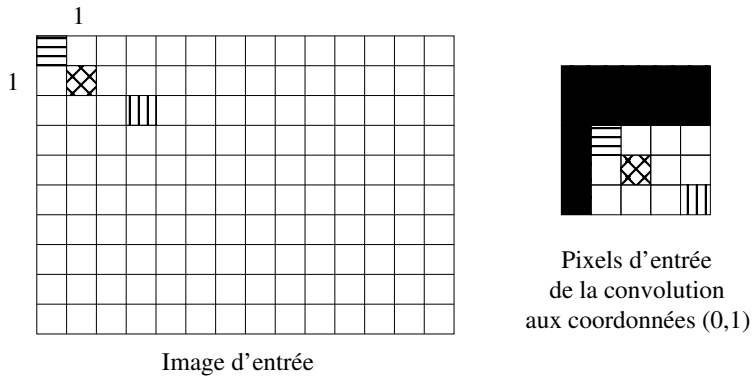

FIGURE 4.9 – **Effets de bord de l'algorithme de convolution 2D** - Les pixels hors de l'image dans la fenêtre de convolution sont ici en noir. Exemple pour un noyau de convolution de 5x5 pixels

l'image d'entrée. Sur une image de 320×240 pixels, une convolution avec un noyau de 9×9 coefficients générera une image de taille 312×232 pixels : on aura rogné l'image de 4 pixels sur chaque bord. Cette solution est attrayante dans le cas d'images de grande résolution en entrée, mais le passage de l'image d'entrée par la cascade de filtres Gaussiens fait que cette solution réduit la perception périphérique du robot dans les basses fréquences spatiales. Au bout des trois premiers filtrages Gaussiens, l'image d'entrée sera passée de 320×240 à 300×220. À la fin du dernier filtrage, l'image de plus basse fréquences spatiales fera 64×44 pixels au lieu de 80×60 pixels, soit une perte de surface de plus de 40%. Ces bandes de basses fréquences étant les plus importantes pour la navigation et la détection d'obstacle, il semble difficile d'envisager un tel défaut dans les algorithmes.

Au lieu de cette solution coûteuse, on préférera réduire la corruption des pixels calculés près des bords. Cette réduction se fera en modifiant les valeurs des pixels invalides, ce qui impactera directement sur la valeur du pixel de sortie. Une maximisation des pixels d'entrée invalides n'est bien sûr pas une idée valable : l'image de sortie serait plus claire sur les bords. Une minimisation de ces pixels entraînerait au contraire un assombrissement des bords. Donner aux pixels invalides la valeur moyenne d'un pixel (127 ou 128 pour un pixel codé sur 8 bits, par exemple), permet de couvrir les images claires et les images sombres : l'impact sur les images très sombres ou très claires sera limité par rapport aux deux solutions présentées.

Cela dit, dans le cas de l'application de vision, il convient de rappeler l'utilisation qui est faite de ces filtrages. En entrée de la cascade de filtres se trouve l'intensité du gradient de l'image de la caméra, généralement sombre, sauf environnement visuel très chargé d'arètes et de points saillants. En sortie, une détection de maxima locaux, permet d'identifier des points les plus saillants de l'image. Un bord trop clair créera artificiellement des faux points saillants au bord de l'image, empêchant la détection de vrais points d'intérêt trop proches. Un bord sombre aura pour effet d'atténuer les valeurs des points du bord. La deuxième solution est préférable, car elle permet de détecter plus facilement des points d'intérêt réels dans l'image.

On choisira par conséquent d'annuler la valeur des pixels invalides.

Les coordonnées du pixel d'entrée étant connues par les IP, il est relativement aisé de détecter les pixels invalides. Les multiplieurs étant rangées par leurs coordonnées respectives au noyau, il est aisé d'annuler le produit quand le pixel d'entrée d'un multiplieur est reconnu invalide. Un multiplexeur suffit à passer à l'additionneur correspondant une valeur nulle plutôt que la sortie du multiplieur.

Cette solution réduit l'effet de bord de l'algorithme, mais le rayon d'inhibition de l'opérateur de recherches de points d'intérêt reste moins efficace que dans les zones mathématiquement justes. Un point saillant sur le bord sera perçu comme moins saillant en sortie de filtrage que s'il se trouvait au centre de l'image. Un point légèrement moins saillant, un peu plus éloigné du bord, pourra alors être détecté comme point d'intérêt alors qu'il est dans le rayon d'inhibition du premier point.

Cette erreur, due au manque d'informations sur l'environnement visuel au delà des bords de l'image, se retrouve dans tous nos déploiements de l'application.

L'erreur maximale générée par cet algorithme est représentée en figure 4.10 : on peut voir les effets de bords sur les étapes successives de filtrage Gaussien. L'image étudiée fait 320×240 pixels, et représente une intensité de gradient de valeur maximale sur toute l'image (quasiment impossible à obtenir en situation réelle). Pour une intensité de gradient de valeur maximale sur toute l'image, l'annulation des pixels au delà des bords de l'image assombrit les bords de l'image résultat. Cette figure permet de visualiser le résultat des filtrages successifs et les conséquences du cumul de ces effets de bords. En

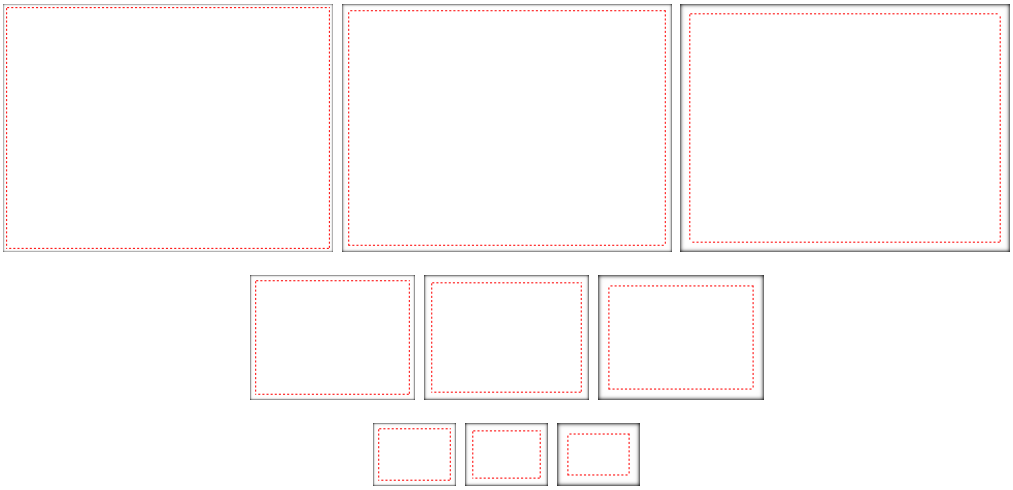

FIGURE 4.10 – **Effets de bord cumulatifs théoriques des convolutions** - Pyramide Gaussienne appliquée à une image blanche de 320×240 pixels. En pointillés rouges, les bords du résultat de convolution parfaite. Au bord des images en noir, l'erreur maximale due aux effets de bord

pointillés rouges, on peut voir les bordures des images générées par des convolutions mathématiquement parfaites. On peut voir que dans les basses fréquences, la méthode approchée permet d'étudier des images beaucoup plus grandes. On voit aussi que la convolution mathématiquement parfaite exclut des pixels dont la valeur estimée a une erreur très faible.

4.5 Différence de Gaussiennes

La différence de Gaussiennes n'étant que la soustraction de flots de pixels deux à deux dans notre cas, l'unité de calcul se résume à un simple opérateur de soustraction. Les pixels de sortie de cette IP sont signés, codés en complément à deux. La difficulté de cette IP réside dans la synchronisation des deux flux de pixels d'entrée. En effet, ils correspondent à l'entrée et la sortie d'une même IP de filtrage Gaussien, ainsi la latence de l'IP de filtrage sépare les deux flux dans le temps. On peut voir l'IP en figure 4.11.

Un registre à décalage permet de retarder le premier flux de pixels d'entrée, jusqu'à ce que les coordonnées du premier flux retardé et du deuxième flux soient identiques. Les coordonnées des pixels des deux flux permettent à la simulation de s'assurer de leur

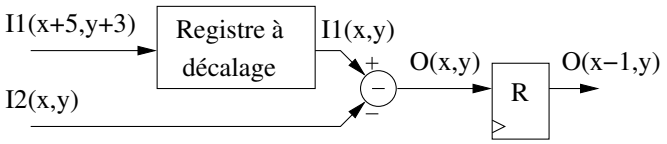

FIGURE 4.11 – **IP de Différence de Gaussiennes** - Soustraction de deux flots de pixels. Un registre à décalage est utilisé pour synchroniser les deux flots

synchronisation, la latence fixe générée par ce registre à décalage pouvant ainsi être réglée de façon sûre. Ainsi, le calcul du pixel de sortie à ces mêmes coordonnées peut être réalisé. Un registre en sortie de l'opérateur de soustraction génère une latence d'un cycle d'horloge (vis-à-vis du deuxième flux de pixels).

4.6 Sous-échantillonnage

Comme pour les IP de gradient et de filtrage Gaussien, un élément important de cette IP est la gestion des données d'entrée de l'opérateur. Pour quatre pixels en entrée de l'opérateur, un seul pixel sera généré en sortie. Cependant, ces quatre pixels ne sont jamais consécutifs dans le flux de pixels d'entrée de l'IP. La gestion des entrées de l'opérateur nécessite par conséquent une attention particulière.

Le sous-échantillonnage est calculé en moyennant des blocs de 2×2 pixels non-recouvrants. Ce mécanisme est décrit en figure 4.12 À partir d'un flot continu de pixels, une FIFO de $W/2$ éléments (W étant la largeur de l'image) permet de stocker les sommes partielles correspondant à la première ligne, puis aux autres lignes paires. Les sommes sont stockées sur un bit de plus que les pixels afin de ne pas réduire la précision du résultat. Cette FIFO sera lue durant l'arrivée dans l'IP des pixels de la deuxième ligne, puis des autres lignes impaires, afin de poursuivre les calculs pour chaque bloc. Ainsi, la FIFO se remplit et se vide toutes les deux lignes de l'image d'entrée. Les blocs de 2×2 pixels sont donc divisé en sous-blocs horizontaux de 2 pixels par ligne. Ces blocs étant non-recouvrant, l'IP utilisera le bit de poids faible des coordonnées X et Y (parité de la colonne/ligne) afin de savoir quel élément du bloc 2×2 est en entrée de l'IP. Une fois la somme des quatre pixels d'un même bloc calculée, il n'y a plus qu'à supprimer les deux bits de poids faible pour obtenir le calcul de la moyenne, un pixel de sortie ayant la même largeur de bus qu'un pixel d'entrée.

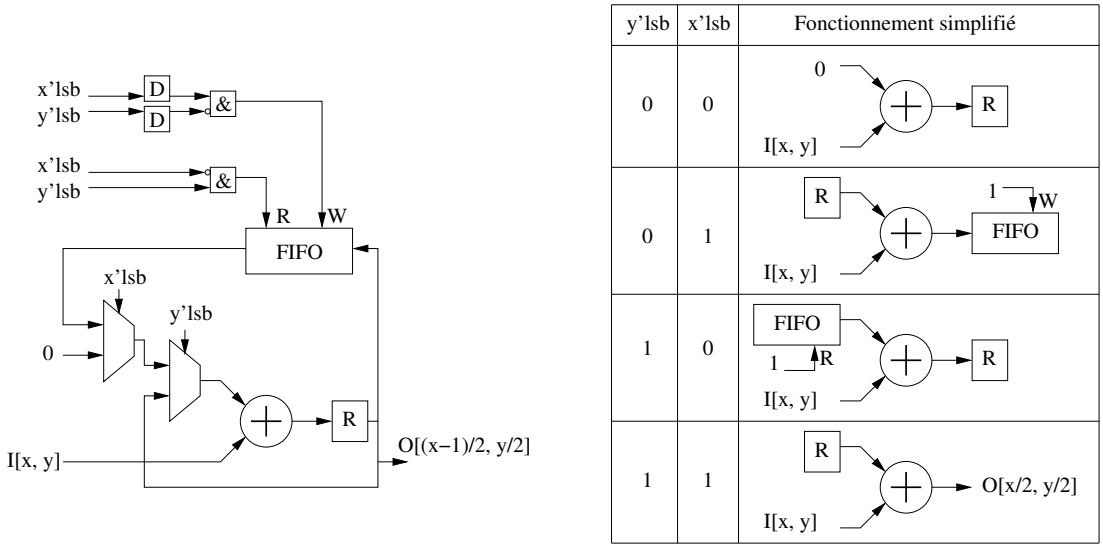

FIGURE 4.12 – **IP de sous-échantillonnage** - Sous-échantillonnage à partir d'un flot de pixels consécutifs. Les parités (bits de poids faible) de x et de y définissent le stade du calcul de l'IP

4.7 Recherche de points d'intérêt

La recherche de point d'intérêt correspond à une détection de maxima locaux dans les images générées par les DoG. Pour chaque pixel, on cherche dans une zone circulaire de rayon R si un autre pixel est supérieur au pixel étudié, afin de déterminer s'il est maximum dans cette zone. L'opérateur se déplace donc sur l'image par fenêtre glissante, de façon similaire à un opérateur de convolution traditionnel (figure 4.4), au détail près que la fenêtre est circulaire. Une fenêtre de pixels de l'image d'entrée est traitée à chaque coup d'horloge. Tous les signaux intermédiaires sont mémorisés dans des structures adaptées, afin que l'IP puisse traiter en parallèle tous les pixels qu'elle recouvre.

L'identification du pixel au centre de la fenêtre comme point d'intérêt est déclenchée par la validation de quatre tests, comme on peut le voir en figure 4.13 :

– La valeur de chacun des pixels de la zone circulaire est comparée à la valeur du pixel central. Si le pixel central est supérieur ou égal à tous les autres pixels de la fenêtre, il correspond donc à un maximum local. Il peut donc être détecté comme point d'intérêt. Voir figure 4.14.

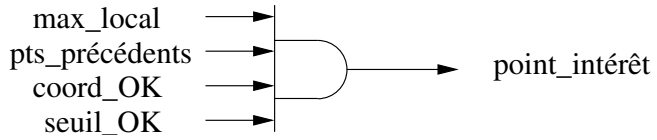

Figure 4.13 – **IP de détection de points d'intérêt : élément décisionnel** - les quatre tests en entrée de cette porte logique "ET" sont décrits dans les prochaines figures

- La non-détection comme point d'intérêt des pixels précédemment testés qui sont recouverts par la fenêtre circulaire de rayon R. Les pixels du demi-disque haut de la zone, et du rayon à gauche du pixel testé, ont déjà été testés par le détecteur. La mémorisation de R lignes de ces résultats permet de savoir si un de ces pixels a déjà été détecté comme point d'intérêt. Si un ou plus des pixels précédents dans cette zone circulaire est un point d'intérêt, le pixel central ne peut pas être un point d'intérêt. Ce mécanisme garantit une distance minimale entre deux points d'intérêt. Voir figure 4.15.
- Le seuil de bruit γ permet de filtrer les maxima locaux de trop faible valeur. Si le pixel central est supérieur ou égal à ce seuil, alors il pourra être détecté comme point d'intérêt. Ainsi, un maximum local dans une zone trop monotone de l'image de la caméra ne pourra pas être étudié par le réseau de neurones, ce qui lui évite de se baser sur des données trop peu pertinentes. Voir figure 4.16.
- Comme dans l'algorithme présenté en 3.1.3, les pixels au delà du bord de l'image sont considérés comme points d'intérêt potentiels. Il faut donc s'assurer que le pixel testé est assez loin du bord pour que tous les pixels dans la fenêtre circulaire soient valides. Ainsi, ce test est effectué uniquement sur les coordonnées du pixel central de la fenêtre. Si la fenêtre ne dépasse pas de l'image, alors le pixel central de la fenêtre peut être détecté comme point d'intérêt. En effet, un pixel trop près du bord a un voisinage partiellement inconnu (hors de l'image), il n'est donc pas possible de s'assurer qu'il est bien maximum local. Voir figure 4.17.

Les résultats de ces quatre tests se présentent sous forme de bits, à '1' si le test est passé. Une porte logique "et" permet ainsi de valider le pixel au centre de la fenêtre comme point d'intérêt. La valeur et les coordonnées de ce pixel sont transmises à l'IP suivante, responsable du tri des points d'intérêt.

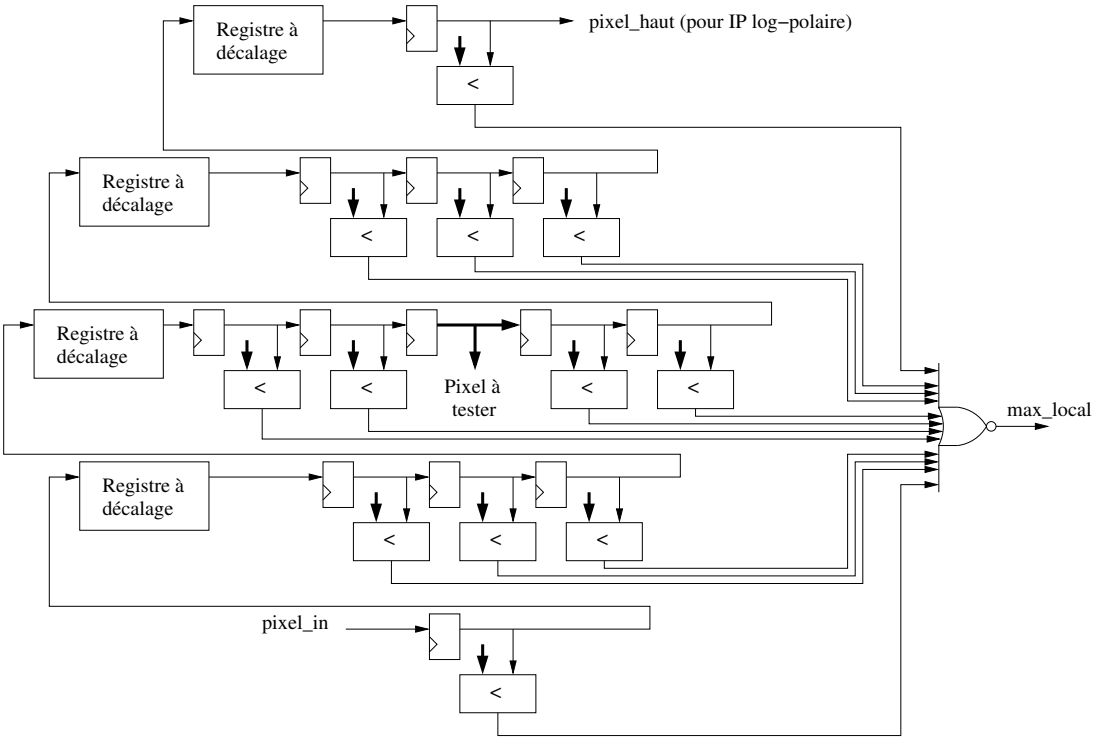

FIGURE 4.14 – **IP de détection de points d'intérêt : détection de maximum local** - exemple pour un rayon de recherche R=2. Les comparateurs testent si les pixels du disque sont supérieurs ou égaux au pixel à tester

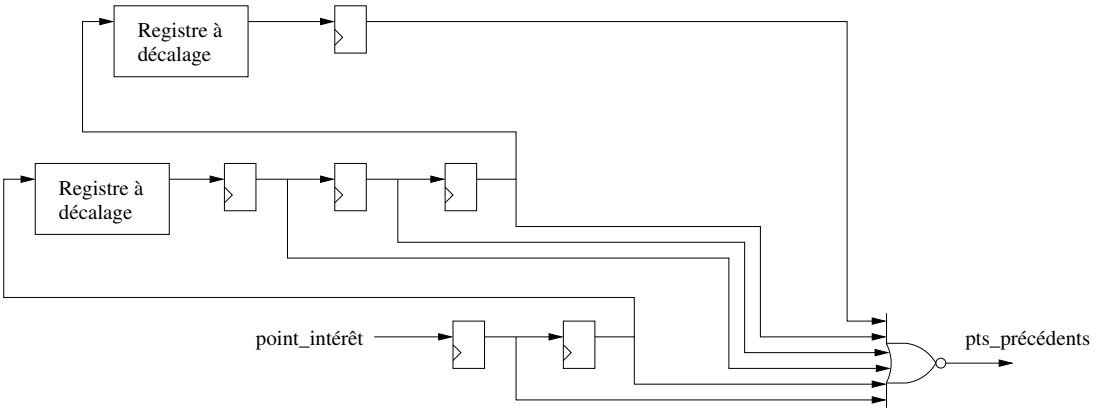

FIGURE 4.15 – **IP de détection de points d'intérêt : inhibition en cas de point d'intérêt précédent trop proche** - exemple pour un rayon de recherche R=2. Ce test se base sur la mémorisation du résultat de l'IP de recherche de points d'intérêt

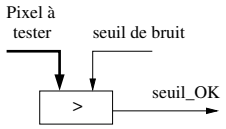

FIGURE 4.16 – **IP de détection de points d'intérêt : seuil de bruit** -

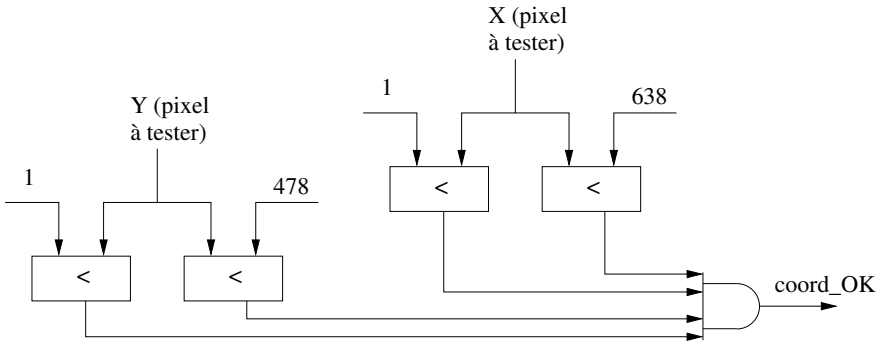

FIGURE 4.17 – **IP de détection de points d'intérêt : inhibition sur les bords de l'image** - exemple pour un rayon de recherche R=2, et une image de 640×480 pixels

Ces quatre tests représentent le cœur fonctionnel de cette IP. On remarque en figure 4.14 que les comparaisons des pixels du disque avec le pixel central se font de façon concurrente. Cette partie de l'IP nécessitera donc d'avoir environ $\pi \times R^2$ comparateurs de deux valeurs de pixels, et $2 \times R$ registres à décalage de dimensions allant de $W - 2$ à $W - 2 \times R$ pour stocker les lignes de pixels. Le deuxième test nécessite de stocker une information de 1 bit (point d'intérêt ou non) pour R lignes de pixels, soit R mémoires de tailles de l'ordre de W bits. Le troisième test n'a besoin que d'un comparateur de deux valeurs de pixels (l'une étant le seuil). Enfin, quatre comparateurs de coordonnées (11 bits pour du 1920×1080 pixels) et une LUT à quatre entrées (pour la fonction logique "et") suffisente pour s'assurer que le pixel central n'est trop près d'aucun des quatre bords.

La grande puissance de cette IP, qui fonctionne en temps-réel sur un flux de pixel, se paie par un coût élevé en opérateurs matériels. Par exemple, pour un rayon de recherche de $R = 20$ et des pixels sur 16 bits, le premier test coûtera environ 1256 comparateurs 16 bits, ce qui nécessite une bonne partie d'un FPGA de taille moyenne actuel. La valeur

de R décroît au fur et à mesure que l'on réduit la résolution, il faudra toutefois deux IP de cette taille pour la haute résolution, les autres résolutions ayant un impact divisé par quatre à chaque sous-échantillonnage. Ce coût en comparateurs augmente en fonction de R de façon quadratique, ce paramètre est donc à régler avec attention.

Dans le cas d'un rayon de recherche assez large, le nombre d'entrées des portes logiques en figures 4.14 et 4.15 peut devenir un frein à la cadence de fonctionnement de l'IP. Afin d'éviter ce désagrément, on regroupe les portes logiques par lignes, une porte supplémentaire permet de regrouper les résultats de ces portes après leur mise en registre.

4.8 Tri de points d'intérêt

Les points d'intérêt sont fournis à l'IP de tri par l'IP de recherche, sous la forme de leurs coordonnées et de la valeur des pixels correspondant. L'IP de tri classe ces ensembles de données comme des blocs indissociables correspondant chacun à un point d'intérêt. La cadence d'entrée des points d'intérêt dans l'IP représente le point le plus délicat de son développement : a priori, à chaque front d'horloge, un nouveau point d'intérêt peut entrer dans l'IP. Il est donc préférable d'avoir une IP capable d'effectuer le tri en un coup d'horloge. Dans cette optique, on dispose d'un avantage de taille : un seul élément à trier peut rentrer dans la liste à chaque coup d'horloge.

La figure 4.18 décrit le fonctionnement de l'IP : la valeur du pixel du nouveau points d'intérêt est comparée à celles de tous les éléments de la liste triée. Chaque point d'intérêt est stocké dans un registre (concaténation de sa valeur et de ses coordonnées). Si un nouveau point d'intérêt entre dans l'IP, il ira se placer immédiatement dans la case adéquate (juste en dessous des points d'intérêt de valeur supérieure ou égale). Tous les pixels de valeur inférieure se décaleront d'un cran vers le bas dans la liste triée. Le mécanisme est simple et modulaire : pour chaque rang de la liste triée, la valeur du pixel du nouveau point d'intérêt est comparée à :

1. la valeur du pixel du point d'intérêt de chaque rang. Si le nouveau pixel a une valeur strictement supérieure à celle du pixel en place, le point d'intérêt en place

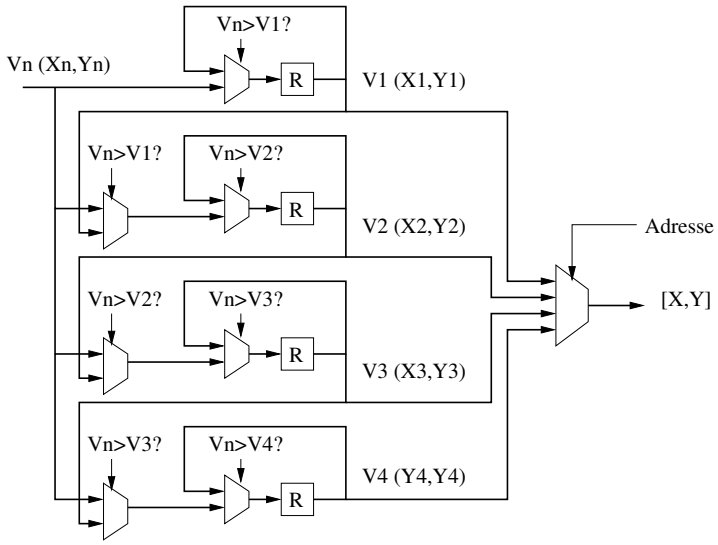

Figure 4.18 – **Tri de points d'intérêt** - Insertion du point d'intérêt de valeur Vn et de coordonnées (Xn,Yn) dans un tableau de 4 points triés

sera remplacé, et sera envoyé au rang de tri directement inférieur (si la taille de la liste le permet ; sinon il sera tout simplement abandonné).

2. la valeur du pixel du point d'intérêt de rang directement supérieur (sauf le premier, n'ayant aucun rang supérieur). Si le nouveau pixel a une valeur strictement supérieure au pixel de rang supérieur, cela signifie que le point d'intérêt de rang supérieur est remplacé, et va prendre la place du point d'intérêt en place.

Un multiplexeur permet ainsi pour chaque rang (sauf le premier) de sélectionner le point d'intérêt entrant ou le point d'intérêt de rang supérieur, en comparant le pixel d'entrée au pixel de rang supérieur. Un autre multiplexeur permet de sélectionner pour chaque rang la valeur en place ou la sortie du multiplexeur précédent (ou le point d'intérêt entrant pour le premier rang), selon que le point d'intérêt devra être remplacé ou non (pixel d'entrée supérieur au pixel en place).

Le fonctionnement en sortie est semblable à une mémoire : une adresse indique l'indice du point dont on souhaite récupérer les coordonnées. Un multiplexeur permet de recopier les coordonnées correspondantes en sortie de l'IP. La valeur des pixels triés n'étant pas utile à la suite des traitements, la sortie de l'IP ne contient que les coordonnées.

4.9 Mise en forme des caractéristiques log-polaires

Les caractéristiques générées par le système de vision permettent au robot d'appréhender son environnement visuel. Le système doit générer pour chaque image fournie par la caméra, une caractéristique pour chaque point d'intérêt.

L'IP de mise en forme des caractéristiques reçoit des informations de l'IP de recherche de points d'intérêt, et de l'IP de tri :

- L'IP de recherche de points d'intérêts lui fournit les pixels qui seront utilisés pour générer les caractéristiques. C'est le signal "pixel_haut" en figure 4.14.
- L'IP de tri lui indique chaque nouveau point d'intérêt détecté dans l'image, son rang dans la liste triée, et ses coordonnées dans l'image.

Un découpage en sous-IP permet la gestion indépendante des caractéristiques, par index dans la liste. Une sous-IP de mise en forme log-polaire sera responsable de la génération de la caractéristique correspondant au point d'intérêt lié à son index. L'IP de tri pouvant remplacer un point d'intérêt par un autre lorsque la liste triée est pleine, une sous-IP doit, à la fin de l'image, avoir généré la caractéristique correspondant au point d'intérêt définitif.

Le schéma en figure 4.19 permet de voir l'organisation de l'IP de mise en forme des caractéristiques, utilisant d'un coté des sorties des IP de recherche et de tri, et de l'autre, permettant une lecture des caractéristiques générées en adressant chaque sous-IP comme une mémoire, une deuxième adresse permettant de sélectionner la sous-IP à lire.

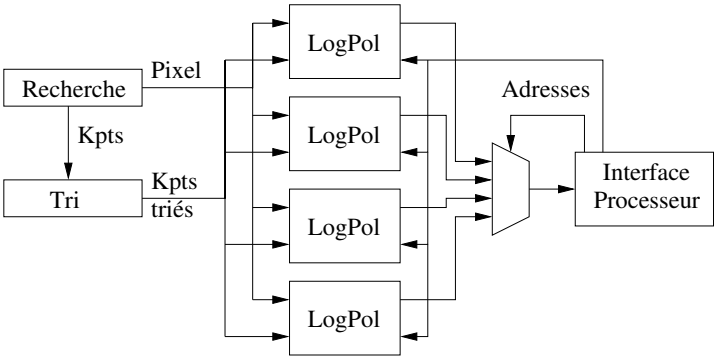

FIGURE 4.19 – **Génération des caractéristiques log-polaires : schéma global** - pour une liste de 4 points d'intérêt

L'interface processeur dépend de la plate-forme sur laquelle le système de vision sera déployé. Dans les tests qui ont été effectués, une interface Avalon a permis à un processeur NIOS II de lire les caractéristiques log-polaires pour les envoyer au réseau de neurones via Ethernet.

Le fonctionnement de chaque sous-IP "logpol" en figure 4.19 est présenté en figure 4.20.

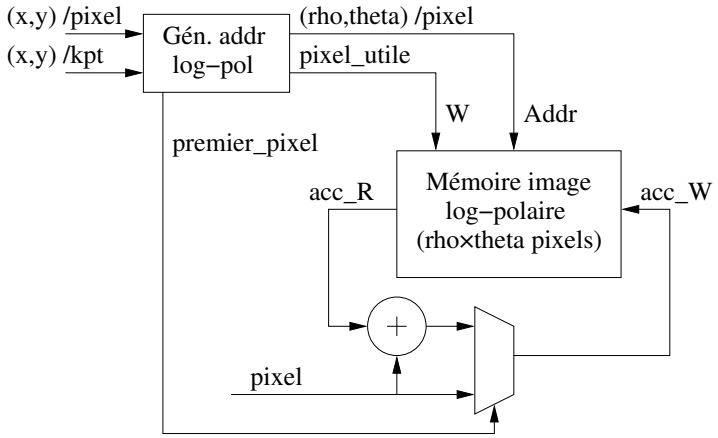

Figure 4.20 – **Génération des caractéristiques log-polaires : pour chaque point d'intérêt** - contenu de chaque bloc "logpol" de la figure 4.19

Les coordonnées de deux pixels sont tout d'abord étudiées : celles du point d'intérêt pour lequel on génère la caractéristique, et celles du pixel d'entrée de l'IP (sortie de l'IP de recherche de points d'intérêt). Le bloc de génération d'adresse permet, à partir de ces deux jeux de coordonnées, de calculer les coordonnées log-polaires du pixel d'entrée, et de tester si ce pixel est dans le voisinage du point d'intérêt, et ainsi activer le cœur de calcul de l'IP. Un signal supplémentaire est généré par ce bloc : il s'agit, pour chaque jeu de coordonnées (ρ, θ), d'identifier le premier pixel de l'image d'entrée à être utilisé. En effet, pour un pixel log-polaire, on calculera la moyenne des pixels correspondant dans l'image d'entrée. La première étape étant de calculer la somme des pixels, on utilise un additionneur, dont on relie la sortie à l'entrée d'une mémoire double port. Un multiplexeur permet de sélectionner la valeur qui sera mise en mémoire, à savoir la valeur du pixel d'entrée si le signal "premier_pixel" est actif, sinon la sortie de l'additionneur.

La lecture des caractéristiques se faisant de façon séquentielle par l'interface processeur, l'opérateur de division permettant le calcul de la moyenne est unique pour chaque IP de mise en forme des caractéristiques, comme on peut le voir en figure 4.19. Le calcul de cette division peut aussi être exécuté logiciellement, sur le processeur embarqué du système. Cette solution permet de réduire la consommation de ressources de cette IP, et de gagner en précision du calcul des caractéristiques en virgule flottante, le réseau de neurones utilisant cette représentation pour ses données.

Un dernier aspect important de la sous-IP de transformation log-polaire est la gestion de la mémoire : cette IP utilise une topologie de type ping-pong, comme on peut le voir en figure 4.21. Ainsi, la lecture des caractéristiques de l'image précédente par le processeur se fait en parallèle de l'écriture des caractéristiques de l'image présente.

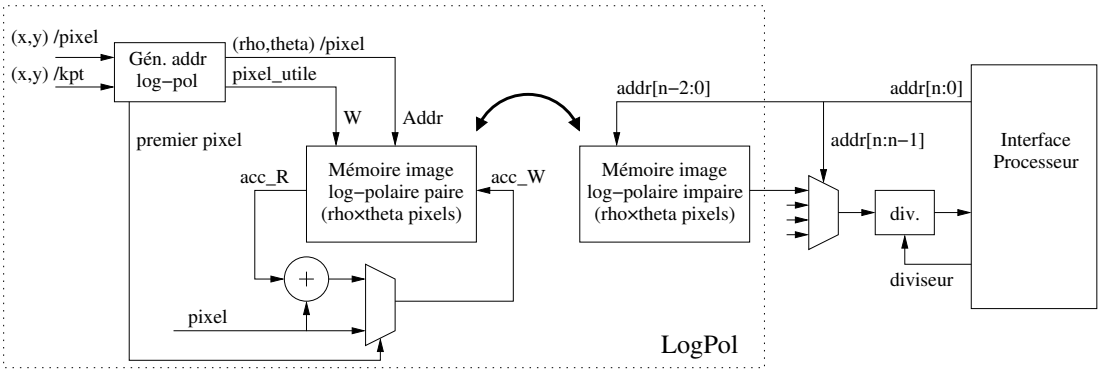

FIGURE 4.21 – **Génération des caractéristiques log-polaires : topologie ping-pong** - la flèche en gras fait apparaître l'utilisation de la topologie ping-pong.

4.10 Conclusion

Ce chapitre a permis de présenter les différents composants matériels réalisés au cours de ce projet, qui permettent d'exécuter la totalité de la chaîne algorithmique du système de vision. Toutefois, la connexion de cet ensemble d'IP à une caméra et au réseau de neurones n'a pas été présenté, car ces parties du système sont dépendantes de l'environnement des IP proposées. La caméra est, à l'heure de la publication de ce document, inadaptée aux besoins du robot, et une solution alternative est recherchée. Concernant la connexion au réseau de neurones, la plate-forme de démonstration

réalisée par Laurent Fiack utilise un processeur NIOS-II pour envoyer par Ethernet les caractéristiques générées par le système. La carte de développement FPGA utilisée pour cette démonstration utilise un FPGA à bas coût, pour valider une version allégée du système complet. Ainsi, la connexion au réseau de neurone n'est elle non plus pas fixée définitivement.

Le choix d'une plate-forme exécutant la totalité de la chaîne algorithmique sous forme d'IP ou déportant certains calculs sur un processeur embarqué reste à faire, en fonction des résultats obtenus au chapitre précédent, et de l'utilisation des ressources FPGA des différentes IP. Il est ainsi nécessaire d'étudier la répartition des ressources FPGA des IP présentées afin de mieux choisir la topologie du système de vision.