Approches SIMD, SMP et MIMD-DM pour la stabilisation 2D d'images en temps réel

SIMD, SMP and MIMD-DM approaches for real-time 2D image stabilization

Jean-Pierre Derutin, Lionel Damez, Fabio Dias et Nicolas Allezard

LASMEA – UMR 6602 du CNRS, Université Blaise Pascal, 24 avenue des Landais – 63177 Aubière
{derutin,damez,dias}@lasmea.univ-bpclermont.fr, nicolas.allezard@cea.fr

Manuscrit reçu le 23 mai 2006

Résumé et mots clés

Nous présentons une méthode de stabilisation de séquence d'images en temps réel, basée sur un modèle de mouvement 2D, avec mise en correspondance par détection et suivi de primitives. Le déplacement estimé entre deux images est filtré afin d'isoler la composante non voulue du mouvement, et finalement utilisé pour corriger les images rendant la séquence stable. Afin de valider l'approche dans un contexte de système temps réel embarqué, l'algorithme a été implanté et testé sur plusieurs plateformes matérielles différentes, permettant l'évaluation des performances selon l'architecture adoptée. Nous montrons ici quelques résultats obtenus, concernant notamment la parallélisation de l'algorithme au moyen des instructions SIMD ALTIVEC, l'adoption d'une architecture symétrique multiprocesseur (SMP) et l'implantation sur une architecture de type MIMD-DM.

Stabilisation 2D d'images, application temps réel, instructions SIMD, architecture SMP, architecture MIMD-DM.

Abstract and key words

We present a real-time image stabilization method, based on a 2D motion model, and exploiting different levels of parallelism in its implementation. This stabilization method is decomposed into three parts. First, the image matching is determined by a feature-based technique. In the second part, the motion between consecutive frames is estimated and filtered to extract the unwanted motion component. Finally, these component is used to correct (warp) the images, resulting in a stable sequence. To validate our stabilization approach in a real-time on-board system context, the algorithm was implemented and tested over different hardware platforms, allowing a performance evaluation in function of the adopted architecture. In this paper, we present some results concerning the parallel implementation of the algorithm, using the SIMD ALTIVEC instructions set, a symmetric multi-processor architecture (SMP) and a MIMD-DM architecture.

2D image stabilization, real-time application, SIMD instructions, SMP, MIMD-DM.

1. Introduction

La stabilisation électronique d'images joue un rôle important dans plusieurs systèmes de vision artificielle, parmi ceux-ci la télé-opération, la robotique mobile, la reconstruction de scène, la compression vidéo, la détection d'objets mouvants, et beaucoup d'autres. Chacune de ces applications a ses spécificités, et le concept de « stable » peut donc varier selon les besoins et les contraintes de chaque application.

Nous nous intéressons au cas général d'une caméra embarquée sur un système mobile, et fixée à celui-ci de façon rigide. Cette configuration est fréquemment présente dans des systèmes de télé-opération ou d'aide à la conduite. La séquence d'images issue de cette caméra contient des informations concernant le déplacement du véhicule dans son environnement. Ce déplacement peut être scindé en deux composantes : celle due aux mouvements commandés du véhicule, et une deuxième composante issue des mouvements parasites (non-commandés) subis par la caméra (rugosité du terrain, vibrations, etc.). Cette composante parasite peut, selon son amplitude, nuire de façon très significative la visualisation et la compréhension des images, soit par un observateur/opérateur humain, soit par un système de vision artificielle.

Dans ce cas, la stabilisation de la séquence consiste dans l'élimination ou l'atténuation de la composante non voulue du mouvement, tout en conservant la composante issue des mouvements commandés. On parle donc de stabilisation sélective.

Si la stabilisation électronique d'images est un sujet qui a déjà mérité un grand nombre de travaux et publications, l'approche architecturale permettant à de tels systèmes de fonctionner en temps réel, tout en respectant les contraintes d'un dispositif embarqué, reste un sujet peu discuté. Il est néanmoins important de valider l'approche algorithmique dans des conditions expérimentales réalistes, notamment au niveau temporel. Pour cela, ce type d'algorithme nécessite une architecture plus ou moins spécialisée et/ou capable de traitement parallèle.

Nous abordons la problématique d'Adéquation Algorithme Architecture en se basant sur une approche *« Commodity Off The Shelf »* (COTS), utilisant des composants informatiques standards. Dans un premier temps, nous avons validé la précision et la robustesse de notre algorithme sur une architecture séquentielle, et dans un second temps nous avons effectué son implantation temps réel en utilisant les parallélismes potentiels des structures COTS.

Dans la section 2, nous décrivons brièvement les différentes méthodes de stabilisation existantes, et les sous modules de traitement qui en font généralement partie. Dans la section 3, nous présentons une description plus détaillée de l'approche de stabilisation développée dans le cadre de ce travail. La section 4 décrit la méthodologie employée pour effectuer la qualification de l'algorithme. La section 5 est consacrée à la présentation des différentes architectures matérielles proposées. Dans la section 6, nous montrons comment les fonctions ont été parallélisées et finalement, dans la section 7, nous abordons quelques résultats obtenus concernant les différentes implantations matérielles.

2. La stabilisation électronique d'images

Depuis quelques années plusieurs méthodes de stabilisation électronique d'images ont été proposées. Ces méthodes peuvent être classifiées dans trois familles principales, selon le modèle d'estimation du mouvement adopté : les méthodes 2D ou planaires [9], les méthodes 3D [5] et les méthodes 2,5D [15]. En effet, les algorithmes de stabilisation sont constitués d'une séquence de traitements de différents niveaux, appliqués sur les images successives de la séquence vidéo. Généralement, trois modules de traitement sont présents :

1. la mise en correspondance entre les images ;
2. l'estimation du déplacement global (au moyen du modèle de mouvement choisi) ;
3. la correction/compensation des mouvements afin d'obtenir une séquence stable.

2.1. Mise en correspondance

L'objectif de la mise en correspondance est de calculer le déplacement d'un point ou région de la scène réelle dans le plan image 2D. Ce déplacement dans le plan image est la projection du déplacement 3D de l'objet dans la scène observée. Deux techniques sont couramment employées : le calcul du flot optique [5], et la détection et suivi de primitives visuelles [9], comme dans la méthode présentée en section 3.

Le calcul du flot optique est un outil classique de traitement d'images, qui a déjà été le sujet d'innombrables publications, dont nous citerons [8] et [3]. Même si cette technique a déjà été employée dans le cadre de la stabilisation d'images, elle présente quelques inconvénients, comme par exemple la complexité des calculs demandés, qui peut être relativement élevée, ou la validité du flot optique comme projection du champ de mouvement 3D, discutée dans [13].

Notre option pour la mise en correspondance des images a été la détection et suivi de primitives visuelles. La technique consiste, dans un premier temps, à localiser dans l'image i des régions riches en information visuelle (fort contraste de luminance, coins, bords, etc.), et dans un deuxième temps, à retrouver ces mêmes régions à l'image $i+1$. Pour détecter ces régions riches en informations, appelées primitives, différentes techniques existent. Le détecteur de coins et bords de Harris et Stephens [7] est une des techniques les plus connues. D'autres possibilités sont les opérateurs Laplaciens ou les ondelettes de Harr, technique expliquée dans la prochaine section.

Après avoir détecté les primitives, il faut être capable de les retrouver dans une autre image. Cela est fait à l'aide d'une fonction de corrélation et d'une stratégie de recherche. Les techniques de recherche multi-résolution permettent de réduire le temps de recherche au moyen d'une approche « coarse-to-fine ». Différentes fonctions de corrélation peuvent être employées, depuis les plus classiques (SSD, SAD), jusqu'à d'autres robustes par exemple aux changements d'illumination dans la scène, comme la corrélation normalisée [12]. Dans [10], plusieurs techniques de corrélation sont présentées et comparées.

2.2. Estimation du déplacement

Une fois que les images successives de la séquence ont été mises en correspondance, une deuxième étape du traitement commence. Il s'agit de la détermination des paramètres décrivant le mouvement entre les images. Les paramètres à déterminer dépendent du modèle de mouvement choisi. Les modèles 2D supposent une scène planaire ou presque, c'est-à-dire que tous les points mis en correspondance et utilisés dans l'estimation se trouvent à plus ou moins la même distance de la caméra. Dans ce cas, il y a trois paramètres à estimer : deux translations (horizontale et verticale) et une rotation autour de l'axe optique de la caméra. Un quatrième paramètre peut être introduit, afin de prendre en compte les changements d'échelle dus à l'avancement de la caméra [9].
Les modèles 3D assument généralement que seules les rotations 3D parasites sont significatives. Il suffit donc d'estimer et corriger celles-ci pour stabiliser la séquence. Comme l'effet sur les images dû aux rotations de la caméra est indépendant de la profondeur des points, il est alors possible d'estimer ces rotations, à l'aide par exemple des quaternions [9].
Le modèle 2,5D présenté dans [15] suppose que quelques connaissances *a priori* sur le mouvement de la caméra sont disponibles, et estime trois paramètres globaux du mouvement, plus un paramètre indépendant pour chaque point analysé, lié à sa profondeur (distance par rapport à la caméra). Cela permet de prendre mieux en compte la complexité structurale d'une scène, sans pourtant faire appel à un modèle 3D.

2.3. Compensation des mouvements

Finalement, après l'estimation du mouvement global entre les images, il faut procéder à la correction de sa composante non voulue. Cette dernière étape du traitement est fortement liée à l'application, car c'est celle-ci qui définie quelle partie du mouvement doit être conservée, et quelle partie doit être compensée. Quelques méthodes adoptées sont la compensation totale du mouvement estimée, l'application d'un filtre numérique passe-bas ou inertiel [15], ou l'approximation polynomiale des rotations 3D estimées [5].

3. Notre méthode de stabilisation

3.1. Descriptif général

Nous avons développé [1] [4] une méthode de stabilisation basée sur un modèle de mouvement 2D, avec détection de primitives par ondelettes de Harr. Comme dans [14], le calcul des ondelettes est effectué à partir d'une image transformée (*image intégrale*). Deux autres images de résolution inférieure ($\frac{1}{2}$ et $\frac{1}{4}$) sont aussi calculées à partir de l'image intégrale, afin de permettre la recherche des points correspondants des primitives à partir d'une approche pyramidale sur trois niveaux de résolution. Celle-ci est effectuée en appliquant une fonction de corrélation (SAD – *Sum of Absolute Differences* ou SSD – *Sum of Square Differences*) entre le motif recherché (primitive), et ses potentiels correspondants dans l'image suivante. Une fois obtenus les appariements de points entre deux images successives, les paramètres du modèle 2D (t_x, t_y et θ) sont estimés par la méthode des Moindres Carrés Médians. Cette méthode est robuste aux faux appariements qui peuvent avoir lieu lors de l'étape précédente. Finalement, et afin d'isoler la composante non voulue du mouvement, les paramètres de déplacement sont filtrés par un filtre numérique passe-bas, et ensuite appliqués pour corriger l'image correspondante dans le but de stabiliser la séquence (figure 1).

3.2. Construction de l'image intégrale

Une image intégrale est calculée à partir de chaque image reçue depuis la caméra (256 niveaux de gris, taille paramétrable). Cette image est utilisée pour la détection par ondelettes, mais aussi pour la génération de deux images sous-échantillonnées ($\frac{1}{2}$ et $\frac{1}{4}$).
L'image intégrale est calculée selon l'équation 1. Elle contient à la position (X,Y) la somme de tous les pixels contenus à l'intérieur du rectangle borné par $i(0,0)$ et $i(X,Y)$, où $i(x,y)$ est l'image originale, et $ii(x,y)$ est l'image intégrale :

$$ii(X,Y) = \sum_{\substack{x \leq X \\ y \leq Y}} i(x,y) \tag{1}$$

À partir de l'image intégrale, la somme de tous les pixels contenus dans une zone rectangulaire peut être obtenue en quatre accès à l'image et 3 additions, indépendamment du nombre de pixels dans cette zone (fig. 2 et eq. 2).

$$\sum_{\substack{x_1 \leq x \leq x_2 \\ y_1 \leq y \leq y_2}} i(x,y) = ii(x_2,y_2) + ii(x_1-1,y_1-1) - ii(x_2,y_1-1) - ii(x_1-1,y_2) \tag{2}$$

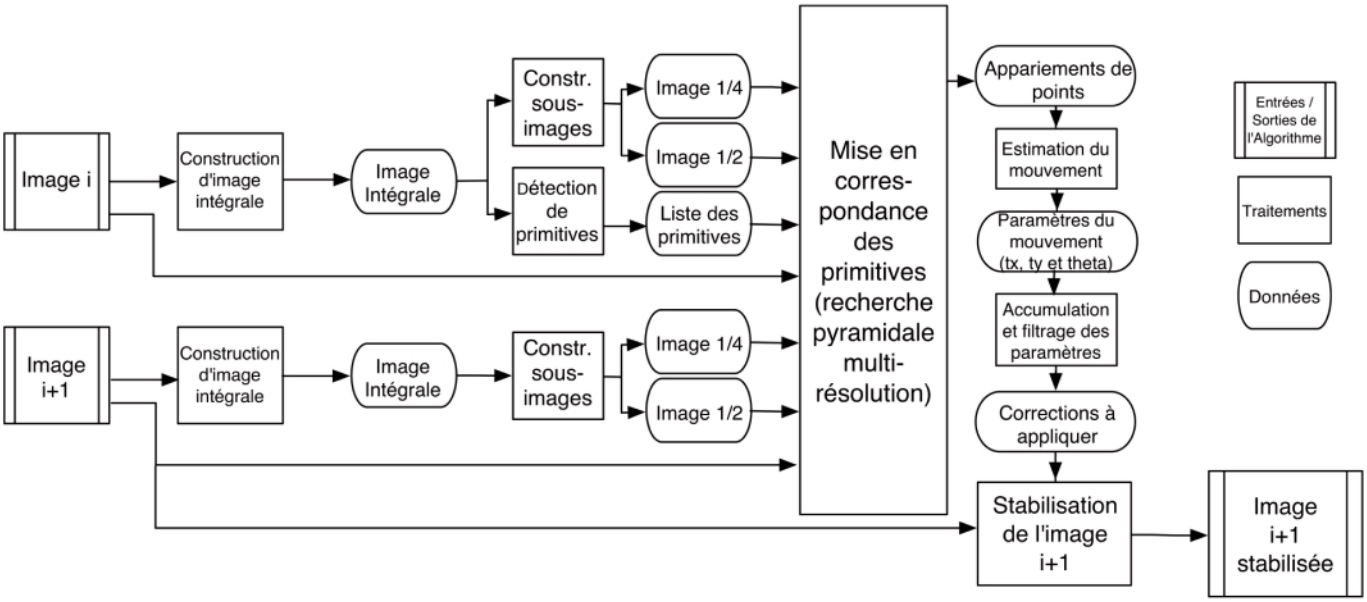

Figure 1. Schéma synoptique d'une itération de l'algorithme de stabilisation.

10	10	9	9	8	7	6	5	6	5
10	10	12	12	11	10	8	6	6	4
9	8	15	14	14	12	7	5	2	1
8	9	16	18	16	14	9	4	2	2
7	9	12	14	12	11	7	5	1	3
8	9	10	12	11	11	8	5	3	2
12	11	13	13	12	10	10	6	4	4
15	14	15	15	13	11	10	7	4	5
15	14	15	15	13	11	10	7	4	5
19	16	15	12	10	9	8	8	5	5

10	20	29	38	46	53	59	64	70	75
20	40	61	82	101	118	132	143	155	164
29	57	93	128	159	190	211	227	241	251
37	74	126	179	224	271	301	321	337	349
44	90	154	221	282	336	373	398	415	430
52	107	181	260	332	397	442	472	492	509
64	130	217	309	393	468	523	559	583	604
79	159	261	368	465	551	616	559	867	713
97	193	312	432	541	639	715	767	801	832
106	228	362	494	613	720	804	864	903	939

Figure 2. Image 10x10 (à gauche) et son image intégrale (à droite).

Afin de réduire le nombre d'opérations effectuées, le calcul de l'image intégrale utilise les formules de récurrence ci-dessous (eqs. 4), où $s(x,y)$ est une valeur intermédiaire (somme accumulée ligne par ligne sur une même colonne) :

$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ ii(x,y) &= ii(x-1,y) + s(x,y) \\ s(x,0) &= i(x,0) \text{ et } ii(0,y) = s(0,y) \end{aligned} \tag{3}$$

Quand appliquées selon une méthode directe, la détection par ondelettes et le calcul d'images sous-échantillonnées demandent un grand nombre de sommes effectuées sur l'image. Avec l'utilisation de l'image intégrale comme image intermédiaire, au lieu de calculer la somme des pixels un à un, nous pouvons calculer directement la somme sur un rectangle de taille quelconque avec seulement 4 accès à l'image intégrale et 3 sommes [14].

3.3. Détection de primitives

Les primitives sont recherchées en appliquant les ondelettes dans une zone de détection de taille $q \times r$. Nous utilisons la moitié supérieure de l'image, ce qui dans une scène d'extérieur permet de détecter les primitives situées à l'horizon. Ces régions sont normalement assez éloignées de la caméra, ce qui permet de respecter au mieux la contrainte de scène planaire imposée par le modèle 2D.

La zone de recherche est divisée en $\frac{n}{3}$ bandes verticales de taille $(3\frac{q}{n}, r)$, où n est le nombre de primitives recherchées fixé par l'utilisateur. La division en bandes permet d'obtenir une bonne distribution spatiale des primitives détectées. Avec par exemple une image de format $T_x \times T_y = 1280 \times 960$, on a $q = 1024$ et $r = 384$, résultant dans des bandes verticales de taille 128×384 pour $n = 24$. Les valeurs q et r représentent les dimensions de la moitié supérieure de l'image, moins une marge de détection de chaque côté. Cette marge sert à empêcher la détection de primitives trop proches du bord de l'image, et qui, dû aux mouvements de la caméra, risquent de ne pas être présentes à l'image suivante, rendant ainsi impossible leur future mise en correspondance.

Chaque bande est balayée par trois types d'ondelettes : une verticale, une horizontale et une diagonale. L'application d'une ondelette consiste dans la convolution d'une région 10×10 de l'image avec un des masques présentés dans la figure 4. La valeur retournée représente le gradient de luminance du motif dans une direction donnée. Pour chacune des trois ondelettes, le point qui retourne la valeur de gradient la plus élevée est retenu comme primitive pour le suivi (fig. 3).

Le calcul des ondelettes est fortement accéléré par l'utilisation de l'image intégrale. Le calcul d'une ondelette 10×10, sans utiliser l'image intégrale, nécessite 100 accès image et 99 opé-

Figure 3. À gauche, primitives sélectionnées par les ondelettes de Harr sur une séquence d'images réelle. À droite, schéma de la mise en correspondance des primitives entre deux images.

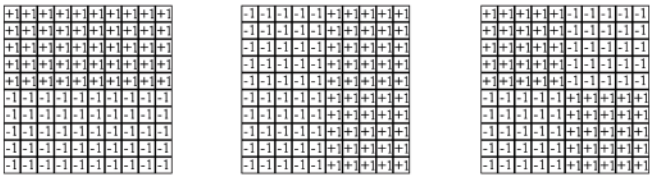

Figure 4. Les masques des trois types d'ondelettes employés. De gauche à droite : ondelettes verticale, horizontale et diagonale.

rations de somme. En utilisant l'image intégrale, une ondelette horizontale ou verticale est calculée en seulement 6 accès à l'image intégrale, 5 additions et 2 multiplications. Une ondelette diagonale est calculée en 9 accès, 8 additions et 1 multiplication, pour n'importe quelle taille d'ondelette.

3.4. Construction des images sous-échantillonnées

Dans cette étape sont générées les images sous-échantillonnées utilisées lors de la recherche pyramidale. Ces images sont obtenues en calculant la luminance moyenne sur une zone carrée de 4 pixels (résolution $\frac{1}{2}$) ou 16 pixels (résolution $\frac{1}{4}$). Au moyen de l'image intégrale, le calcul de chaque pixel des deux images sous-échantillonnées nécessite 3 sommes et une division.

3.5. Mise en correspondance de primitives

Considérant que l'étape de détection a été réalisée sur l'image i, nous allons maintenant rechercher dans l'image $i+1$ les correspondants des n primitives retenues. Tout d'abord, une fenêtre de recherche de taille $2T \times 2T$ est définie autour de la position où la primitive a été détectée (figure 3). Ensuite, la corrélation entre chaque région à l'intérieur de cette fenêtre et le motif retenu comme primitive à l'image i est calculée. La région de l'image $i+1$ qui présente un meilleur score de corrélation est considérée comme étant la correspondante à cette primitive. Nous avons donc un appariement de points entre les deux images consécutives. Cette opération est réalisée pour chacune des n primitives détectées.

Afin de diminuer le nombre d'opérations à réaliser, la recherche est faite au moyen d'une approche multi-résolution. Nous utilisons d'abord l'image sous-échantillonnée $\frac{1}{4}$, et la recherche est faite à l'intérieur d'une fenêtre de taille $\frac{T}{2} \times \frac{T}{2}$. Cela nous donne une première estimation de la position du correspondant.

À partir de cette estimation, une deuxième recherche est effectuée sur l'image sous-échantillonnée $\frac{1}{2}$. Cette fois-ci, la recherche est faite à l'intérieur d'une fenêtre 3×3 autour de la position estimée précédemment. Une deuxième estimation est donc obtenue, plus précise que la première. Finalement, une dernière recherche est faite sur l'image originale, avec une précision sous-pixelique ($\frac{1}{8}$ de pixel). Les intensités sous-pixeliques sont estimées via une interpolation bilinéaire sur une fenêtre 2×2.

3.6. Estimation du mouvement

En possession des n appariements de points entre les images i et $i+1$, les paramètres du modelée 2D, décrivant le mouvement d'une image à l'autre, peuvent être estimés. Il est supposé que le déplacement peut être modélisé par une matrice de transformation homogène (eq. 4), constituée d'une rotation autour de l'axe optique (θ), d'une translation horizontale (t_x) et d'une translation verticale (t_y). Le déplacement d'un point (x, y) entre deux images est donc décrit par le système ci-dessous, où (x', y') est la position du point (x, y) après déplacement :

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & t_x \\ -\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4}$$

Les trois paramètres inconnus de la matrice (θ, t_x et t_y) sont estimés en appliquant ce modèle aux n appariements de points retrouvés à l'étape de mise en correspondance. Notons p_1 le nuage de points d'intérêt détectés à l'image i, et p_2 le nuage formé par leurs correspondants trouvés à l'image $i+1$. Les deux nuages de points sont centrés par rapport à leurs barycentres respectifs, donnant deux nuages centrés p'_1 et p'_2, liés par une rotation autour de l'origine : $p'_2 = R.p'_1$. Il faut donc minimiser le critère :

$$S = \sum_i \left\| p'_{2i} - R \cdot p'_{1i} \right\|^2 \tag{5}$$

La minimisation du critère S donne le paramètre θ, et une fois en possession de l'angle de rotation les deux paramètres de translation sont déduits à partir du mouvement des barycentres des deux nuages. Dans les équations ci-dessous (eqs. 6), (b_{1x}, b_{1y}) et (b_{2x}, b_{2y}) sont les coordonnées du barycentre des nuages p_1 et p_2 respectivement :

$$\begin{aligned} t_x &= b_{2x} - b_{1x} \cdot \cos\theta - b_{1y} \cdot \sin\theta \\ t_y &= b_{2y} + b_{1x} \cdot \sin\theta - b_{1y} \cdot \cos\theta \end{aligned} \tag{6}$$

L'estimation des paramètres du mouvement est réalisée au moyen d'un filtrage par les moindres carrés médians, technique robuste qui tolère jusqu'à 50 % de données aberrantes.

3.7. Accumulation et filtrage des paramètres

Les paramètres estimés sont accumulés avec ceux calculés précédemment, afin de trouver le déplacement global de la caméra au long de la séquence. Les valeurs θ, t_x et t_y calculées précédemment sont appliquées pour obtenir la matrice de transformation de l'image i à l'image $i+1$ (eq. 7). Cette matrice est multipliée par la matrice Mat_i, contenant l'accumulation des déplacements depuis le début de la séquence (eq. 8). Nous obtenons donc la matrice de transformation Mat_{i+1}, qui décrit le déplacement de l'image $i+1$ par rapport à un repère de référence.

$$Mat_{i\to i+1} = \begin{pmatrix} \cos\theta & \sin\theta & t_x \\ -\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

$$Mat_{i+1} = Mat_{i\to i+1} \cdot Mat_i \tag{8}$$

Un filtre linéaire du premier ordre est appliqué indépendamment à chaque paramètre (eqs. 10). Les coefficients des trois filtres peuvent être réglés par l'utilisateur (K_x, K_y, K_θ). Cela permet d'obtenir une stabilisation sur mesure selon les contraintes de l'application. Le découplage des filtres permet encore d'avoir différents niveaux de stabilisation pour la rotation et les translations.

$$\begin{aligned} t_x[i+1] &= Mat_{i+1}[1,3] * (1-K_x) \\ t_y[i+1] &= Mat_{i+1}[2,3] * (1-K_y) \\ \theta[i+1] &= acos(Mat_{i+1}[1,1]) * (1-K_\theta) \end{aligned} \tag{9}$$

Finalement, les valeurs filtrées sont utilisées pour obtenir la matrice de déplacement inverse, qui est donc appliquée à l'image $i+1$ afin de la stabiliser, c'est-à-dire, de la ramener vers le repère de référence (fig. 5). Ce repère de référence est dynamique, et essaie d'accompagner les mouvements commandés de la camera, selon les coefficients des filtres choisis par l'utilisateur.

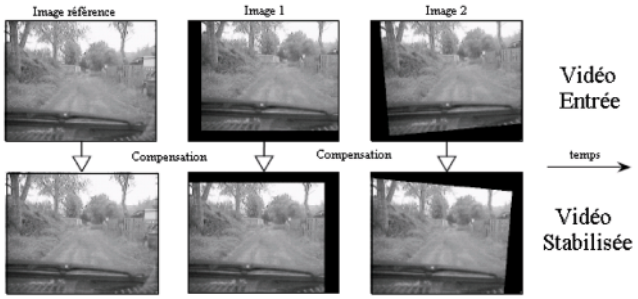

Figure 5. Séquence d'images synthétiques instable (en haut) et séquence après stabilisation (en bas).

4. Qualification de la stabilisation

Afin de valider l'algorithme de stabilisation expliqué précédemment, nous avons conçu une méthodologie de qualification. Ceci s'avère nécessaire car une simple vérification visuelle du fonctionnement de l'algorithme ne nous permet pas d'obtenir des données quantitatives indispensables. Ces informations permettent de mieux comprendre et prévoir le comportement du système face à diverses situations d'application, sans avoir à tester le dispositif de façon exhaustive.

La qualification de la stabilisation d'images est un sujet peu discuté dans la communauté scientifique. Du fait que le concept de stabilité peut varier en fonction du domaine d'application, qualifier un tel algorithme sans tenir compte du cadre d'utilisation s'avère une tâche complexe. Les différents aspects pouvant intervenir dans la performance d'un tel système sont :

- la précision d'estimation du mouvement de la caméra ;
- le déplacement maximal d'un motif visuel pouvant être mesuré par le système ;
- le confort de visualisation de la séquence résultante ;
- le temps de calcul nécessaire pour traiter chaque image de la séquence.

Dans [2], une méthode de qualification indirecte est proposée. L'idée est d'appliquer la stabilisation comme prétraitement pour un autre algorithme de vision, comme par exemple un dispositif de détection et suivi de cibles. Cette approche permet de vérifier la validation du dispositif face à une application réelle, en mesurant l'apport de la stabilisation sur les performances d'une autre application. Par contre, elle ne permet pas d'obtenir des données quantitatives.

Une méthode d'obtention de la précision d'estimation est présentée dans [9]. La somme des différences entre chaque pixel de deux images consécutives préalablement stabilisées est calculée. En supposant une scène statique, si la stabilisation est parfaite les deux images sont identiques, et la différence est nulle. L'inconvénient majeur de cette méthode est sa forte sensibilité au contenu des images, notamment le contraste. De plus, même si les résultats obtenus sont quantifiables, ils ne sont pas directement liés aux paramètres impliqués lors du processus de stabilisation.

Pour obtenir la précision réelle d'estimation du mouvement de la caméra, il est indispensable d'avoir une vérité de terrain sur le déplacement de celle-ci, ce qui n'est pas possible à obtenir pour des séquences réelles. Nous avons donc choisi de qualifier notre approche en utilisant des séquences d'images synthétiques, dont nous maîtrisons le modèle de déplacement et le contenu des images. Ainsi, il est possible de comparer les déplacements réels avec ceux estimées par l'algorithme, obtenant donc l'erreur d'estimation.

Ces séquences synthétiques de test sont générées à partir d'une image de référence et d'un modèle de déplacement, choisis en fonction du domaine d'application visé. L'image choisie

comme référence doit contenir les mêmes textures et objets qui seront retrouvés lors d'une utilisation sur le terrain : arbres, ciel, route, horizon. Une image réelle prise par une caméra embarquée peut être utilisée (fig. 5). De même, le modèle de déplacement utilisé doit être cohérent avec les mouvements auxquels la caméra est généralement soumise sur le terrain. Nous utilisons un modèle d'évolution du mouvement extrait d'une séquence réelle, prise à partir d'un véhicule roulant sur un terrain accidentée.

Des transformations 2D rigides sont appliquées successivement à l'image de référence, au moyen d'une matrice de déplacement contenant les paramètres du mouvement définis dans le modèle. Chaque transformation génère une nouvelle image de la séquence, simulant ainsi une caméra qui se déplace parallèlement au plan optique, et observant une scène statique. Ces mouvements sont planaires et homogènes sur toute l'image, respectant parfaitement les contraintes du modèle planaire 2D qui sont assumées dans l'algorithme. Le fonctionnement et la précision de la stabilisation peuvent donc être observés, indépendamment de la validité ou non des hypothèses du modèle planaire 2D sous certaines conditions réelles (cette validité ne peut être vérifiée qu'au cas par cas, selon le type de scène traitée).

Les tests sur plusieurs séquences synthétiques montrent que l'approche proposée est très précise. Les erreurs d'estimation sont inférieures à 0,2 pixels pour les translations, et inférieures à 0,05 degrés pour les rotation.

Le déplacement maximal pouvant être mesuré par notre système est réglé par l'utilisateur (paramètre T), ayant une incidence directe sur le temps de calcul. Le plus loin on cherche, le plus de temps on passera à chercher. Ce couplage évident entre le déplacement maximal et le temps d'exécution est extrêmement important. En effet, la grandeur à maximiser est la multiplication du déplacement maximal (en pixels) par le nombre d'images traitées par seconde. Cela indique la valeur maximale de la vitesse d'un objet, en pixels par seconde, pour qu'il puisse être suivi par le dispositif. Ainsi, le paramètre T doit être réglé en fonction du temps de calcul et de son influence sur celui-ci.

Si l'estimation du déplacement de la caméra est précise, le confort de visualisation obtenu dépend entièrement du filtrage réalisé. Grâce aux filtres paramétrables, notre méthode de stabilisation est capable de s'adapter à un très large éventail de situations, permettant le rendu d'une séquence stabilisée de facile visualisation et compréhension.

Les temps de calcul de l'algorithme, étant très liés à l'architecture matérielle employée, sont présentés et analysés dans les prochaines sections.

5. Présentation des architectures

La chaîne de traitement décrite en section 3 a été développée sur une machine de type PC, équipée d'un processeur AMD Athlon XP 1700. Après l'évaluation et la validation de l'efficacité de l'algorithme, celui-ci a été parallélisé et porté sur des machines PowerMac d'Apple, à architecture symétrique biprocesseur à mémoire partagée. Ces machines possèdent deux processeurs MPC7455 de Motorola (G4), ou deux PowerPC 970 d'IBM (G5). Leur système d'exploitation est MacOS X. Les architectures présentées disposent de capacités de traitements parallèles exploitables à trois niveaux :

- Au niveau de l'exécution sur un seul processeur, avec des mécanismes d'exécution super scalaire, et avec le jeux d'instructions SIMD Altivec ;
- Au niveau d'une machine (noeud), possédant deux processeurs pouvant travailler simultanément et partageant la mémoire (SMP) ;
- Au niveau d'un cluster de machines interconnectées, constituant une architecture à mémoire distribuée (MIMD-DM).

5.1. Jeu d'instruction SIMD

Ce type d'extension est rencontré dans la plupart des microprocesseurs actuels : MMX, SSE et SSE2 pour les processeurs Intel, 3DNOW! pour les processeurs AMD, MDMX pour les processeurs MIPS et VIS pour les processeurs SPARC. Ces extensions du jeu d'instructions sont toutes fondées sur deux principes :

- Le premier est d'offrir des capacités de traitement de type SIMD : effectuer simultanément une même opération arithmétique ou logique sur un ensemble de données avec une seule instruction (figure 6).

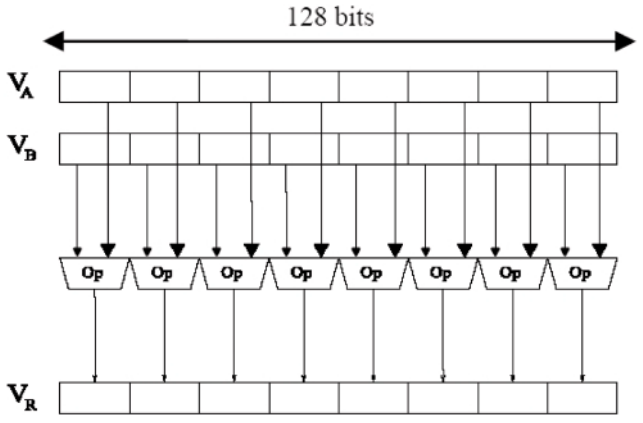

Figure 6. Instruction C Altivec à deux opérandes : **VR = vec_add(VA, VB)**.

- Le deuxième principe est de fournir un jeu d'instructions fortement inspiré des architectures DSP : arithmétique saturée, opérateurs câblés, opérateurs de conversion de type, etc. (figure 7).

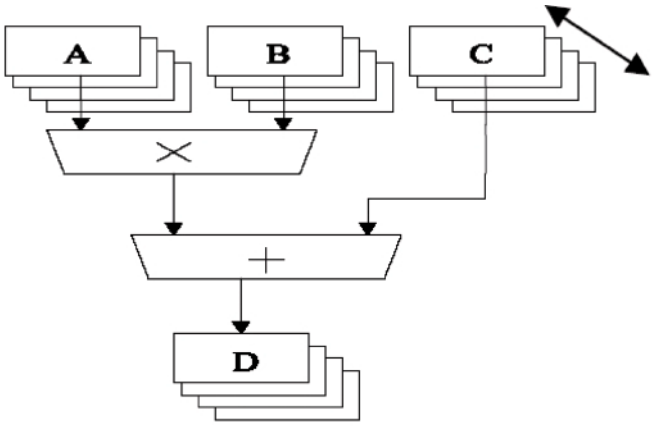

Figure 7. Instruction C Altivec de type DSP : **VD = vec_madd(VA, VB, VC)**.

Ces instructions s'appliquent sur des registres de taille fixe (128 bits pour Altivec), mais dont le nombre d'éléments traités simultanément peut varier : les éléments 32 bits sont traités par lots de 4, ceux de 16 bits sont traités par lots de 8 et ceux de 8 bits par lots de 16.

L'utilisation des jeux d'instructions SIMD permet de mettre en oeuvre un parallélisme à grain fin particulièrement adapté aux traitements réguliers. Dans ce cas, il est démontré que si le ratio « opérations de calcul/opérations mémoire » d'un traitement est suffisamment élevé, il est possible d'obtenir des accélérations quasi-linéaires (4, 8 ou 16) pour les calculs sur des nombres entiers, voire des accélérations sur-linéaires dans le cas des calculs en virgule flottante. Les performances obtenues grâce à ces jeux d'instructions sont discutées dans [11] et [6].

Cependant, la parallélisation SIMD limite son champ d'application aux traitements réguliers. Un autre inconvénient est qu'elle oblige à rester à un faible niveau d'abstraction. En conséquence, des parties entières du programme qu'on souhaite accélérer sont à réécrire.

5.2. Architecture SMP

Le second niveau de parallélisme réside dans l'exploitation de deux processeurs communicant via une mémoire partagée. Cette architecture permet de mettre en oeuvre un parallélisme gros grain, en répartissant des tâches ou en découpant des blocs de données entre les processeurs.

Le système d'exploitation MacOS X est conçu pour répartir l'exécution des différentes tâches sur plusieurs processeurs, gérant l'allocation des tâches en cours d'exécution par planification préemptive. L'exécution en parallèle se fait alors de façon complètement transparente pour l'utilisateur. Cependant, afin que le système exécute des tâches appartenant à une même application sur différents processeurs, il faut séparer ces tâches explicitement, sous forme de processus légers (threads). Dans le cas de la chaîne de stabilisation, deux processus légers sont donc crées et exécutés en concurrence, permettant alors au système de repartir leur exécution entre les ressources processeur disponibles.

La création de ces processus est possible en faisant des appels système avec la bibliothèque de fonctions standard « pthread ». Cette bibliothèque définit des primitives de création et fusion de processus, ainsi que des mécanismes de verrous pour permettre de synchroniser et mettre en place des mécanismes d'exclusion mutuelle. À la différence des jeux d'instructions SIMD, le niveau d'abstraction élevé permet une implantation parallèle sans avoir nécessairement à réécrire intégralement les traitements à paralléliser.

5.3. Architecture MIMD-DM

L'architecture utilisée est composée de 14 machines PowerMac G5, interconnectées avec un switch en réseau Ethernet Gigabit, permettant de mettre en oeuvre un parallélisme gros grain. Comme chaque machine comprend 2 processeurs en mémoire partagée, l'architecture décrite totalise 28 processeurs avec une organisation mémoire hétérogène (NUMA).

Dans un souci d'homogénéité, toutes les communications se font par passage de message, aussi bien entre processeurs de noeuds distincts qu'entre processeurs d'une même machine. Ces communications sont effectuées avec la librairie MPI. Implantée sur un grand nombre de plateformes, cette bibliothèque apporte plus d'une centaine de fonctions de niveau d'abstraction élevé. Elle définit notamment un ensemble de primitives de communication point à point et d'opérations collectives, simplifiant la mise en oeuvre de structures parallèles sur machines MIMD-DM.

6. Implantation parallèle

Afin de proposer un schéma de parallélisation exploitant au mieux les structures architecturales définies précédemment, tout en favorisant d'une part la rapidité de portage et d'autre part les meilleurs résultats en terme de temps d'exécution, la démarche suivante a été mise en place.

6.1. Analyse de l'algorithme séquentiel

Les différentes étapes de la version séquentielle de l'application ont été analysées selon deux critères : ratio « traitements/opération mémoire » et volume de données traitées (tableau 1).

L'importance relative des différentes étapes de l'algorithme par rapport au temps total d'exécution de la boucle de stabilisation en mode séquentiel est indiquée sur le tableau 2. En se basant sur ces analyses, il est possible de concentrer l'effort de parallélisation sur les étapes permettant à priori un gain plus notable. Nous pouvons constater que les étapes *step_1*, *step_2* et *step_4* sont les plus consommatrices de puissance de calcul. Grâce à l'utilisation de l'image intégrale pour le calcul des ondelettes, l'étape *step_3* a des temps d'exécution négligeables par rapport aux autres étapes de l'algorithme.

Il apparaît que *step_4* est l'étape qui a le temps d'exécution le plus important. Cependant, avec l'augmentation de la taille de l'image, l'importance relative du calcul des images intermédiaires (*step_1* et *step_2*) augmente. Un mode d'exécution parallèle de ces trois étapes est donc souhaitable, afin d'observer un gain de performance intéressant. Nous avons choisi d'implanter les quatre premières étapes en parallèle. Les deux dernières, ne présentant pas une complexité importante (< 2 %), sont réalisées en mode séquentiel. L'étape *step_3* est parallélisée non pas pour obtenir des gains temporels, car ces temps d'exécution sont déjà assez faibles, mais afin de faciliter la parallélisation, évitant une opération de *merge-split* entre les étapes *step_2* et *step_4*.

6.2. Description de l'implantation parallèle

Il est rappelé que l'architecture SMP, tout comme l'architecture de type grappe, offre deux niveaux de parallélisme : de données avec Altivec (mode SIMD), et de tâches et/ou données sur plusieurs processeurs. Nous avons fait le choix d'utiliser des approches d'implantation les plus proches possibles pour les architectures matérielles SMP et MIMD-DM. Les deux approches conservent néanmoins de fortes différences structurelles. Sur l'architecture SMP, les deux processus légers travaillant sur des structures de données séparées, sont créés au sein d'une même application. Pour l'architecture MIMD-DM, nous utilisons le paradigme SPMD (Single Program Multiple Data), où sont exécutées autant d'instances de l'application que de processeurs.

Le schéma d'implantation des étapes *step_1* à *step_4* en mode SMP et MIMD-DM est essentiellement issu de la distribution des données à traiter (images, puis listes de points) entre les p processeurs disponibles. Le partitionnement des données, avec un recouvrement des images produites lors de *step_1* et *step_2*, permet de minimiser les communications entre processeurs.

Les étapes *step_1* et *step_2* sont effectuées par chaque processeur p sur un bloc de l'image i correspondant à la zone de détection, entouré d'une marge $2T + k$, nécessaire au suivi. Chacun de ces blocs a une taille $(\frac{q}{p} + 2T + k,\ r + 2T + k)$ (voir section 3.3). Le coefficient k dépend de la taille du motif de corrélation

Tableau 1. Caractéristiques des différentes étapes de l'application.

Étapes	Parallélisme intrinsèque	Ratio traitements/ opération mémoire	Volume de données traitées
step_1 : Calcul de limage intégrale	données	faible	élevé
step_2 : Sous-échantillonnage des images	tâches et données	faible	élevé
step_3 : Détection des primitives	tâches et données	faible	faible
step_4 : Suivi de primitives par corrélation	tâches et données	très élevé	très élevé
step_5 : Estimation du mouvement	séquentiel	faible	faible
step_6 : Filtrage du mouvement	séquentiel	faible	très faible

Tableau 2. Importance relative des différentes étapes en fonction du format vidéo et de la distance de recherche.

Format vidéo	**320 × 240**	**640 × 480**	**1280 × 960**	**2560 × 1920**	**5120 × 3840**
valeur de T	15	30	60	120	240
step_1 et *step_2*	1%	3%	6%	9%	10%
step_4	97%	95%	93%	90%	90%
step_5 et *step_6*	2%	2%	1%	1%	0%

et du sous-échantillonnage. Chacun de ces blocs peut contenir une ou plusieurs bandes de détection.

La phase *step_3* est effectuée par chaque processeur p sur une zone de détection de taille $(\frac{q}{p}, r)$, et chaque processeur produit une liste de $\frac{n}{p}$ primitives qui seront suivies lors de *step_4*. Le nombre de bandes verticales est fonction du nombre de points que chaque processeur doit détecter, à raison de 1 bande pour 3 points. Cette repartition est faite aussi équitablement que possible entre les processeurs, avec au plus une différence de 1 point entre le processeur le plus chargé et celui le moins chargé. Par exemple pour $n = 84$, avec 8 processeurs, 4 processeurs détectent 11 points et 4 autres détectent 10 points. Avec 28 processeurs, ils ont tous exactement 3 points à traiter. Il est donc possible d'obtenir une répartition spatiale différente des points détectés en fonction du nombre de points et de processeurs. Pour reprendre l'exemple, avec $n = 84$ points, pour $p = 1$ ou $p = 28$ on obtient 28 bandes de 3 points chacune. Cependant, pour $p = 8$, chaque processeur traite un bloc contenant 4 bandes, ce qui fait au total 32 bandes.

L'étape de suivi (*step_4*) est l'étape la plus coûteuse en temps de traitement. Elle permet le cumul de deux niveaux de parallélisme dans son implémentation parallèle. D'une part en mode MIMD-DM ou SMP, chaque processeur cherche les $\frac{n}{p}$ motifs qu'il a précédemment détecté. D'autre part, les résultats de recherche de maximum de corrélation sont obtenus en utilisant les instructions SIMD Altivec. De cette manière, chaque processeur est capable de traiter jusqu'à 16 pixels en une seule opération, ce qui réduit d'autant le nombre d'opérations. Cette fonction de corrélation existe en deux versions différentes. Une fonction plus simple, de type SAD, est appliquée lors des deux premiers étages de la recherche multi résolution, travaillant sur des nombres de type entier. Sur le dernier étage, pour obtenir une précision sous-pixelique, une fonction de type SSD est appliquée, travaillant sur des nombres à virgule flottante. Les dimensions des primitives recherchées sont choisies de manière à permettre d'optimiser le nombre de traitements SIMD et les accès aux données. Ainsi, puisque les pixels peuvent être lus en mémoire par blocs de 16, des motifs de taille multiple de 16 sont échantillonnés dans l'image i puis recherchés dans l'image $i + 1$. La SAD est calculée en ôtant les minimums de luminance des maximums, sur des nombre entiers 8 bits non signés. Ceci permet d'obtenir un parallélisme de 16, sans la perte de précision à cause du bit de signe qui serait engendrée par une soustraction directe entre les valeurs. La fonction SSD est effectuée sur des nombres de type flottant 32 bits, permettant un parallélisme théorique de 4. La complexité de cette opération est due principalement aux nombreuses opérations de conversion de type (vecteur de nombres entiers vers des vecteurs de nombres flottants), et aux interpolations bilinéaires, plus qu'à l'opération de SSD proprement dite.

L'étape de fusion des résultats (merge) n'existe que dans la version parallèle : après la phase de suivi, toutes les listes d'appariement de points obtenues par chaque processeur sont concaténées en une seule liste. En mode MIMD-DM, ceci implique la collecte des résultats par un seul processeur (*mpi_gather*). L'essentiel des communications entre les processeurs intervient pendant cette opération. En mode SMP, les deux processus sont aussi joints dans cette étape.

Finalement, les étapes d'estimation du mouvement et filtrage (*step_5* et *step_6*), implantées en séquentiel, permettent d'ache-

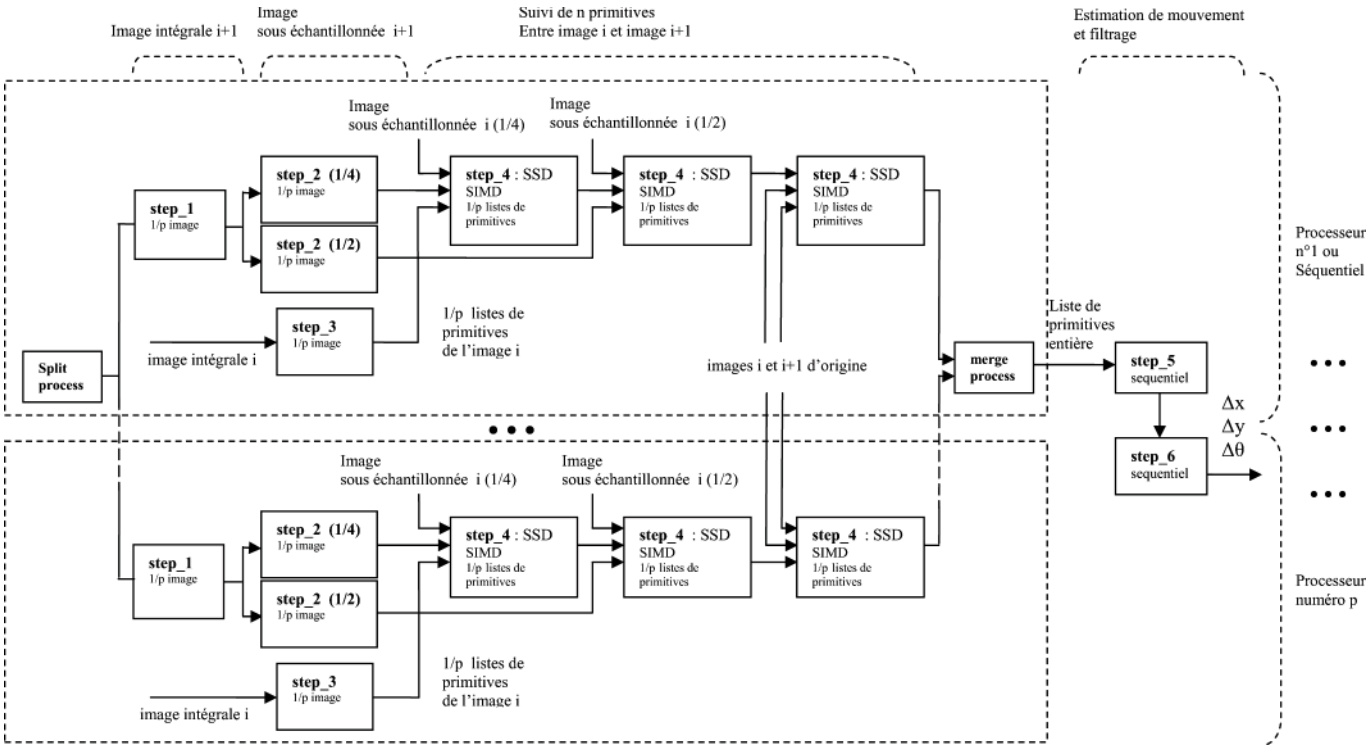

Figure 8. Schéma de parallélisation de l'algorithme.

ver le traitement et réaliser la stabilisation. Un schéma de la parallélisation de l'application sur plusieurs processeurs est montré en figure 8.

7. Résultats

Les performances temporelles de la stabilisation ont été évaluées avec 2 séries de mesures. La première série (tableaux 4 et 5) a été effectuée sur les trois configurations de machine décrites précédemment (tableau 3): [*arch_1*] de type compatible PC, et [*arch_2*] et [*arch_3*] de type POWERPC. Dans cette série, le traitement est appliqué à une séquence d'images enregistrée, format 320x240, avec un nombre réduit de primitives recherchées ($n = 24$, $T = 15$, paramètres *bench_1*). La deuxième série de mesures (tableaux 6 à 13) a été effectuée sur [*arch_3*], montrant les temps de traitement de chaque étape de l'algorithme en fonction du nombre de processeurs utilisés. Chaque paire de tableaux correspond à un format d'image, allant de 640x480 (*bench_2*, tableaux 6 et 7) à 5120x3840 (*bench_5*, tableaux 12 et 13). L'algorithme est paramétré pour rechercher un nombre plus important de primitives ($n = 84$), à une distance T proche de 5 % de la largeur des images (de $T = 30$ pixels pour *bench_2* à $T = 240$ pixels pour *bench_5*). Dans les tableaux 4 à 13 sont indiqués:

- Les temps d'exécution des étapes séquentielles et parallélisées;
- Le temps d'exécution d'une boucle de stabilisation complète (de *step_1* à *step_6*);
- Pour l'implémentation SMP, les temps de création et d'arrêt des processus légers;
- Pour l'implémentation MPI, les temps de communications entre processeurs.

Notre cluster ne disposant pas d'un bus vidéo de bande passante suffisante, les vidéos sur lesquels l'algorithme de stabilisation est appliqué proviennent des mémoires de masse reparties sur chaque noeud. Les temps d'acquisition et d'affichage ne sont donc pas comptabilisés. Les mesures correspondent au temps écoulé entre deux appels de fonctions UNIX *get_time()*, moyennées sur 1000 itérations de l'algorithme. À côté du temps d'exécution de chaque étape est indiquée entre parenthèse l'accélération obtenue, d'une part entre la version séquentielle et l'implantation SIMD, d'autre part entre l'implantation SMP ou MPI et l'implantation SIMD.

Tableau 3. Caractéristiques des systèmes utilisés.

Plateforme	**[*Arch_1*] Athlon XP 1700+**	**[*Arch_2*] PowerMac G4**	**[*Arch_3*] PowerMac G5**
Système	Windows XP	Mac OS 10.3	Mac OS 10.3
Compilateur	gcc 3.2	gcc 3.3	gcc 3.3 + lam MPI 7.1.1
Nombre de μP	1	2	de 2 à 28
μProcesseur	Athlon XP	MPC7455	PowerPC970
Fréquence	1,47 GHz	1 GHz	2 GHz
Mémoire vive	512 MB	512 MB	1 GB
Cache L1 **L2** **L3**	128 KB 256 KB	64 KB 256 KB 1 MB	64 KB 512 KB

Tableau 4. Performances temporelles de [arch_1] et [arch_2] – images 320x240 et paramètres bench_1 ($n = 24$ primitives, $T = 15$ pixels).

Etape	[arch_1] séquentiel	[arch_2] séquentiel	[arch_2] SIMD	[arch_2] SIMD + SMP ($2\mu P$)	[arch_2] SIMD + MPI ($2\mu P$)
step_1 et *step_2* (ms)	1,1	1,5	1,5	1,3 (1,2)	1,5 (1)
step_3 (ms)	0,3	0,5	0,5	0,4	0,3
step_4 (ms)	18,8	32,7	2,8 (11,4)	1,5 (1,9)	1,4 (2)
step_5 et *step_6* (ms)	1,4	1,8	1,8	1,8	1,8
Comm./thread overhead (ms)	-	-	-	0,9	0,7
Total (*step_1* à *step_6*) (ms)	21,1	36,5	6,6 (5,5)	5,8 (1,1)	5,6 (1,2)
Accélération Totale				**6,3**	**6,5**

Tableau 5. Performances temporelles de [arch_3] – images 320x240 et paramètres bench_1 ($n = 24$ primitives, $T = 15$ pixels).

Etape	[arch_3] séquentiel	[arch_3] SIMD	[arch_3] SIMD + SMP ($2\mu P$)	[arch_3] SIMD + MPI ($2\mu P$)
step_1 et *step_2* (ms)	0,7	0,7	0,5 (1,4)	0,5 (1,4)
step_3 (ms)	0,3	0,3	0,2	0,1
step_4 (ms)	18,5	1,5 (12,3)	0,8 (1,9)	0,7 (2,1)
step_5 et *step_6* (ms)	0,8	0,8	0,8	0,8
Comm./thread overhead (ms)	-	-	0,6	0
Total (*step_1* à *step_6*) (ms)	20,3	3,3 (6,2)	3,0 (1,1)	2,2 (1,5)
Accélération Totale			**6,8**	**9,2**

Tableau 6. Performances temporelles de [arch_3] – images 640x480 et paramètres bench_2 ($n = 84$, $T = 30$).

Etape	séquentiel	SIMD	SIMD + SMP ($2\mu P$)	SIMD + MPI ($2\mu P$)
step_1 et *step_2* (ms)	2,7	2,7	1,7 (1,6)	1,7 (1,6)
step_3 (ms)	0,4	0,4	0,3	0,2
step_4 (ms)	78,2	7,1 (11,1)	3,7 (1,9)	3,6 (2,0)
step_5 et *step_6* (ms)	1,1	1,1	1,1	1,1
Comm./thread overhead (ms)	-	-	0,9	0,1
Total (*step_1* à *step_6*) (ms)	82,5	11,4 (7,2)	7,6 (1,5)	6,7 (1,7)
Accélération Totale			**10,9**	**12,3**

Tableau 7. Performances temporelles de [arch_3] – images 640x480 et paramètres bench_2 ($n = 84$, $T = 30$).

Etape	SIMD + MPI (4μP)	SIMD + MPI (8μP)	SIMD + MPI (16μP)	SIMD + MPI (28μP)
step_1 et *step_2* (ms)	1,2 (2,3)	1,0 (2,7)	0,8 (3,4)	0,8 (3,4)
step_3 (ms)	0,1	0,1	0,1	0
step_4 (ms)	1,8 (3,9)	1,0 (7,1)	0,6 (11,8)	0,4 (17,8)
step_5 et *step_6* (ms)	1,1	1,1	1,1	1,1
Comm./ thread overhead (ms)	0,6	0,9	1,4	2
Total (*step_1* à *step_6*)(ms)	4,8 (2,3)	4,0 (2,9)	3,9 (2,9)	4,2 (2,7)
Accélération Totale	**17,2**	**20,6**	**21,2**	**19,6**

Tableau 8. Performances temporelles de [arch_3] – images 1280x960 et paramètres bench_3 ($n = 84$, $T = 60$).

Etape	séquentiel	SIMD	SIMD + SMP (2μP)	SIMD + MPI (2μP)
step_1 et *step_2*(ms)	9	9	5,9 (1,5)	5,7 (1,6)
step_3(ms)	0,6	0,6	0,4	0,4
step_4(ms)	137,6	14,8 (9,3)	7,5 (2,0)	7,5 (2,0)
step_5 et *step_6*(ms)	1,1	1,1	1,1	1,1
Comm./ thread overhead (ms)	-	-	0,9	0
Total (*step_1* à *step_6*)(ms)	148,4	25,6 (5,8)	15,7 (1,6)	14,7 (1,7)
Accélération Totale			**9,5**	**10,1**

Tableau 9. Performances temporelles de [arch_3] – images 1280x960 et paramètres bench_3 ($n = 84$, $T = 60$).

Etape	SIMD + MPI (4μP)	SIMD + MPI (8μP)	SIMD + MPI (16μP)	SIMD + MPI (28μP)
step_1 et *step_2*(ms)	4,0 (2,3)	3,0(3,0)	2,5 (3,6)	2,3 (3,9)
step_3(ms)	0,2	0,1	0,1	0
step_4(ms)	3,8 (3,9)	2,0 (7,4)	1,2 (12,3)	0,7 (21,1)
step_5 et *step_6*(ms)	1	1,1	1,1	1
Comm./ thread overhead (ms)	0,7	0,9	1,4	2,2
Total (*step_1* à *step_6*)(ms)	9,7 (2,6)	7,1 (3,6)	6,2 (4,1)	6,1 (4,2)
Accélération Totale	**15,3**	**20,9**	**23,9**	**24.3**

Tableau 10. Performances temporelles de [arch_3] - images 2560x1920 et paramètres bench_4 ($n = 84$, $T = 120$).

Etape	séquentiel	SIMD	SIMD + SMP (2μP)	SIMD + MPI (2μP)
step_1 et *step_2*(ms)	34,2	32,7	20,4 (1,6)	20,2 (1,6)
step_3(ms)	1,4	1,4	0,6	0,4
step_4(ms)	357,1	42,5 (8,4)	21,6 (2,0)	21,8 (2,0)
step_5 et *step_6*(ms)	1,1	1,1	1,1	1,1
Comm./ thread overhead(ms)	-	-	0,9	0,1
Total (*step_1* à *step_6*)(ms)	393,8	77,7 (5,1)	44,6 (1,7)	43,8 (1,8)
Accélération Totale			**8,8**	**9**

Tableau 11. Performances temporelles de [arch_3] – images 2560x1920 et paramètres bench_4 ($n = 84$, $T = 120$).

Etape	SIMD + MPI (4μP)	SIMD + MPI (8μP)	SIMD + MPI (16μP)	SIMD + MPI (28μP)
step_1 et *step_2*(ms)	12,9 (2,5)	9,4 (3,5)	7,7 (4,3)	6,8 (4,8)
step_3(ms)	0,2	0,2	0,1	0,1
step_4(ms)	11,1 (3,8)	5,9 (7,2)	3,5 (12,1)	2,0 (21,3)
step_5 et *step_6*(ms)	1,1	1,1	1,1	1,1
Comm./ thread overhead (ms)	0,7	1	1,7	2,2
Total (*step_1* à *step_6*)(ms)	25,9 (3,0)	17,3 (4,5)	13,8 (5,6)	11,9 (6,5)
Accélération Totale	**15,2**	**22,8**	**28,5**	**33,1**

Tableau 12. Performances temporelles de [arch_3] – images 5120x3840 et paramètres bench_5 ($n = 84$, $T = 240$).

Etape	sequentiel	SIMD	SIMD + SMP (2μP)	SIMD + MPI (2μP)
step_1 et *step_2*(ms)	124	123,9	74,8 (1,7)	72,0 (1,7)
step_3(ms)	1,9	1,9	0,6	0,5
step_4(ms)	1226	152,3 (8,1)	77,9 (2,0)	80,0 (1,9)
step_5 et *step_6*(ms)	1,1	1,1	1,1	1,1
Comm./ thread overhead (ms)	-	-	0,9	0,4
Total (*step_1* à *step_6*)(ms)	1353,1	279,2 (4,8)	155,2 (1,8)	154,0 (1,8)
Accélération Totale			**8,7**	**8,8**

Tableau 13. Performances temporelles de [arch_3] – images 5120x3840 et paramètres bench_5 ($n = 84$, $T = 240$).

Etape	SIMD + MPI (4μP)	SIMD + MPI (8μP)	SIMD + MPI (16μP)	SIMD + MPI (28μP)
step_1 et *step_2*(ms)	44,9 (2,8)	32,4 (3,8)	24,7 (5,1)	21,4 (5,8)
step_3(ms)	0,3	0,2	0,1	0,1
step_4(ms)	39,7 (3,8)	21,7 (7,0)	11,9 (12,8)	6,2 (24,6)
step_5 et *step_6*(ms)	1,1	1,1	1,1	1,1
Comm./ thread overhead (ms)	0,8	0,9	2	4,5
Total (*step_1* à *step_6*)(ms)	87,1 (3,2)	56,0 (5,0)	39,7 (7,0)	32,9 (8,5)
Accélération Totale	**15,5**	**24,2**	**34,1**	**41,1**

7.1. Implantation SIMD

Il est difficile de déterminer une accélération théorique à partir de l'implantation de *step_4*, les calculs de corrélation (SAD et SSD) étant effectués aussi bien sur des nombres entiers 8 bits (accélération théorique de 16) que sur des nombres flottants (accélération théorique de 4). De plus, le modèle d'exécution des instructions utilisées n'est pas seulement de type SIMD, mais aussi de type DSP, comme expliqué dans la section 5.1 et illustré dans la figure 7.

Les instructions SIMD de calcul sur des nombres flottants sont plus efficaces que leur équivalent scalaire. Dans notre implantation, la proportion du nombre d'opérations sur des flottants par rapport au nombre d'opérations sur des entiers diminue avec l'augmentation de la fenêtre de recherche, ce qui fait varier l'efficacité de notre implantation avec T. On observe en effet pour la plateforme [*arch_3*] une accélération allant de 12,3 sur *bench_1* ($T = 15$) à 8,1 sur *bench_5* ($T = 240$). Néanmoins, du point de vue général, on observe une accélération très satisfaisante : pour [*arch_3*] et *bench_3*, l'accélération de *step_4* est de 9,3 et le temps de boucle est inférieur à 30 ms.

7.2. Implantation SMP

L'accélération due au SMP est proche de la valeur attendue pour *step_1*, *step_2*, *step_3* et *step_4*. Pour *bench_1*, l'accélération SIMD + SMP est de seulement 1.1, car les coûts de création et suppression des processus sont peu compensés par la diminution des temps de traitements. Les résultats sont meilleurs sur les autres séries de mesures, avec des formats d'image plus importants et un plus grand nombre de primitives détectées et suivies. Les accélérations dues au mode SMP varient entre 1,5 pour *bench_2* et 1,8 pour *bench_5*. Du point de vue de l'application, l'implantation SIMD+SMP permet à [*arch_3*] de diminuer le temps d'exécution sur *bench_3* (images 1280x960) en dessous de 20 ms, avec une accélération globale de 9,5.

7.3. Implantation MPI

Les performances de l'implantation MPI avec $p = 2$ processeurs sont particulièrement proches de celles de l'implantation SMP, utilisant la même approche de parallélisation. Cependant, les temps de communications de l'implantation MPI sont inférieurs au temps de création et destruction des processus légers dans l'implantation SMP. L'accélération des étapes *step_1* et *step_2* augmente de plus en plus faiblement avec le nombre de processeur p. Ceci s'explique par la présence du coefficient $2T + k$, indépendant de p, dans l'expression de la taille des blocs à traiter ($\frac{q}{p} + 2T + k,\ r + 2T + k$). D'autre part, les temps supplémentaires de synchronisation et de communication augmentent avec le nombre de processeurs p. Pour *bench_4* et *bench_5* (tableaux 10 à 13), indépendamment du nombre de processeurs, les temps de traitement restent suffisamment élevés pour compenser les coûts de communications. Par contre, pour *bench_2* et *bench_3*, avec des dimensions d'image moins importantes, les temps de communication à 28 processeurs (environ 2 ms) deviennent trop importants pour être compensés par l'accélération des temps de traitement. Pour finir, l'accélération de *step_4* due à l'implantation MPI augmente de manière quasi-linéaire avec p.

8. Conclusion et perspectives

Ce papier présente une méthode de stabilisation 2D efficace, avec une précision de calcul des déplacements proche de $\frac{1}{5}$ de pixel. De plus, l'originalité de la détection par ondelettes de Harr permet un allégement important des calculs, qui allié aux efforts d'une implantation architecturale adaptée donne des performances temporelles très efficaces. Cela permet d'envisager l'utilisation de cette méthode comme étape de prétraitement dans une chaîne d'algorithmes de vision. Plusieurs architectures parallèles ont été évaluées. L'utilisation du jeu d'instructions SIMD donne des très bons résultats. L'implantation avec MPI s'est montrée aussi performante qu'en utilisant explicitement la mémoire partagée de l'architecture SMP.
Cette étude a été effectuée d'une part comme étude préliminaire d'algorithme pour la restauration de films anciens, travaillant sur des résolutions très élevées, et d'autre part pour la validation d'une approche SPMD pour une future implantation de l'application sur une architecture de type SOC multi-processeurs, utilisant un réseau de processeurs homogènes.

Références

[1] Nicolas ALLEZARD, Lionel DAMEZ, and Jean-Pierre DÉRUTIN, Real time image stabilization by visual features matching. Rapport de recherche, LASMEA – Université Blaise Pascal, Mar 2003.

[2] S. BALARSKIRSKY and R. CHELLAPPA, Performance characterization of image stabilization algorithms. Rapport de recherche, Center for Automation Research – University of Maryland, 1996.

[3] J. BARRON, D. FLEET, S. BEAUCHEMIN, and T. BURKITT, Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43-77, 1994.

[4] F. DIAS REAL DE OLIVEIRA, *Analyse, développement et qualification d'un algorithme de stabilisation électronique d'images*. Thèse de master, Université Blaise Pascal, 2004.

[5] Z. DURIC and A. ROSENFELD, Shooting a smooth video with a shaky camera. *Machine Vision and Applications*, 13(5-6):303-313, 2003.

[6] J. FALCOU and J. SEROT, E.v.e., an object oriented simd library. In *International Conference and Computation Science – ICCS'2004*, pages 323-330, 2004.

[7] C. HARRIS and M. STEPHENS. A combined corner and edge detector. In *Proceeding of the 4th Alvey Vision Conference*, pages 147-151, 1988.

[8] B. HORN and B. SCHUNCK, Determining optical flow. *Artificial Intelligence*, 17:185-204, 1981.

[9] C. MORIMOTO, *Electronic Digital Stabilization: Design and Evaluation, with Applications*. Thèse de doctorat, University of Maryland, 1997.

[10] H. POURREZA, M. RAHMATI, and F. BEHAZIN, Weighted multiple bit-plane matching, a simple and efficient matching criterion for electronic digital image stabilizer application. In *6th International Conference on Signal Processing*, volume 2, pages 957-960, 2002.

[11] J. SEBOT, Impact des extensions simd sur les performances d'applications multimedia. *Technique et Science Informatiques*, 21(2):185-204, Mar 2002.

[12] D. TSAI, C. LIN, and J. CHEN, The evaluation of normalized cross correlations for defect detection. *Pattern Recognition Letters*, 24:2525-2535, 2003.

[13] A. VERRI and T. POGGIO, Motion field and optical flow: Qualitative properties. In *IEEE Trans. Pattern Analysis and Machine Intelligence*, volume 11(8), 1989.

[14] Paul VIOLA and Michael JONES, Rapid object detection using a boosted cascade of simple features. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, 2001.

[15] Z. ZHU, G. XU, Y. YANG, and J. JIN, Camera stabilisation based on 2.5d motion estimation and inertial motion filtering. In *International Conference on Intelligent Vehicles*, 1998.

Jean-Pierre **Derutin**

Professeur à l'université Blaise Pascal de Clermont-Ferrand, habilité à diriger des recherches en 1993 dans le thème « Architecture des systèmes de perception artificielle ». Ses centres d'intérêts concerne les méthodes et outils de prototypage rapide pour l'aide à la parallélisation dans le domaine de la vision artificielle ainsi que les méthodes de prototypage hardware pour la conception de SoC en utilisant des réseaux paramétrables de processeurs homogènes communicants. Les domaines applicatifs visés concernent les véhicules intelligents.

Lionel **Damez**

Né à Clermont-ferrand en 1977. Titulaire d'un diplome d'ingenieur en genie éléctrique en 2001 et du DEA « Vision pour la Robotique » de l'Université Blaise Pascal en 2002, il est actuellement en thèse au LASMEA, sous la direction de Jean Pierre Dérutin. Ses principaux travaux de recherche concernent l'implementation d'architectures embarquées, basées sur des structures multiprocesseur homogènes avec un reseau de communication embarqué (NoC), ainsi que le portage et la parallelisation d'applications de vision sur ces architectures.

Fabio **Dias**

Né à Sâo Paulo en 1980, il est actuellement en thèse au LASMEA depuis l'année 2004. Il a obtenu un diplôme d'ingénieur électricien à l'Université de Campinas, Brésil, en 2003, ainsi qu'un DEA en « Vision pour la Robotique », à l'Université Blaise Pascal à Clermont-Ferrand, en 2004. Ses axes principaux de recherche sont les caméras intelligentes, les architectures embarquées et les méthodologies d'implémentation d'algorithmes de vision dans les architectures hétérogènes.

Nicolas **Allezard**

Docteur de l'université Blaise Pascal de Clermont-Ferrand, est actuellement ingénieur-chercheur au CEA/LIST de Saclay. Ses Centres d'intérêt sont l'apprentissage automatique pour la reconnaissance des formes, les méthodes pour la mise en correspondance de primitives visuelles ainsi que les techniques stéréoscopiques. Les domaines applicatifs visés concernent principalement l'aide au conducteur et les véhicules intelligents.