Suivi d'objet par capteurs visuels et inertiels sur systèmes embarqués

I. Salhi[12] E. Piriou[1] M. Poreba[2] M. Ojail[1] V. Gouet-Brunet[2]

[1] CEA, LIST, F-91191 Gif-sur-Yvette, France.
[2] Univ. Paris-Est, LASTIG MATIS, IGN, ENSG, F-94160 Saint-Mande, France

imane.salhi@cea.fr

Résumé

Cet article présente une amélioration d'un algorithme de suivi de points d'intérêt par la fusion de données inertielles (Inertial Measurement Unit - IMU) et visuelles (caméra) avec pour finalité sa mise en œuvre sur une caméra intelligente embarquée mobile. L'adaptation proposée repose sur un couplage serré qui s'appuie sur l'algorithme KLT assisté par des données inertielles, implémenté sur CPU et accéléré par un GPU. Cette solution de l'état de l'art donne des résultats précis et robustes de suivi de points d'intérêt par une caméra mobile, mais le traitement est effectué sur une machine type PC fixe (x86). Elle ne tient donc pas compte des contraintes inhérentes aux systèmes embarqués telles que la consommation d'énergie, le coût calculatoire et la surface silicium occupée. Dans cet article, l'étude se focalise sur la mise en œuvre de cet algorithme de suivi robuste sur une cible embarquée (Nvidia Tegra X1) afin de répondre aux exigences des systèmes avec de fortes contraintes d'intégration. Les modifications apportées sont évaluées en comparant la qualité du suivi obtenu et les temps d'exécution sur cible embarquée par rapport à ceux de l'approche de référence. Les expériences montrent que même sur une cible embarquée, la précision du suivi n'est pas dégradée.

Mots Clefs

Données visuelles et inertielles, Fusion, Systèmes embarqués, GPU, Suivi de points d'intérêt, Algorithme KLT.

Abstract

This article describes an enhancement of a data fusion IMU/camera point feature tracking algorithm for the purpose of its implementation on an embedded mobile smart camera. The proposed method is based on a KLT (Kanade-Lucas-Tomasi)-based tightly coupled approach assisted by inertial data (IMU). It is implemented on CPU and accelerated by a GPU device. The original solution provides accurate and robust results regarding feature tracking for a moving camera. However, it was only implemented on a desktop machine (x86) and embedded systems constraints, such as power consumption, computational cost, and space requirements were not considered. In this article, we focus on implementing this robust tracking algorithm on an embedded target (Nvidia Tegra X1) to meet the requirements of systems with strong integration constraints. To assess the quality of our embedded implementation of this point tracking algorithm, we compare it to its desktop-class implementation, according to several criteria (quality of tracking and execution time). The experiments show that even on an embedded target, the accuracy of the tracking is not deteriorated.

Keywords

Visual and inertial data, Fusion, Embedded systems, GPU, Feature point tracking, KLT.

1 Introduction

Le suivi d'objet est un sujet particulièrement complexe à aborder dans le cas des systèmes embarqués, tels que les caméras intelligentes embarquées mobiles. Divers algorithmes et méthodes ont été développés afin de fournir une solution efficace et performante soit en utilisant individuellement des données visuelles (caméra) ou des données inertielles/magnétiques (IMU), soit par leur fusion (IMU/caméra). Ce couplage peut être défini comme lâche [7] ou serré [18, 22], comme l'illustre la figure 1.

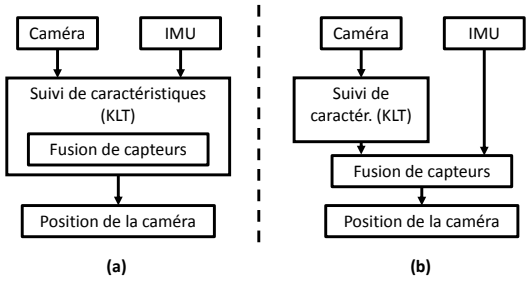

Figure 1: Deux stratégies de couplage IMU/Caméra: (a) couplage serré, (b) couplage lâche [7]

Le couplage serré se caractérise par la fusion des données inertielles et visuelles dès leur intégration dans le système afin de calculer une décision finale. Or, en ce qui concerne le couplage lâche (figure 1-(b)), les données inertielles et

visuelles sont traitées indépendamment. Leur fusion est effectuée au niveau des décisions partielles de ces traitements indépendants afin de produire une décision finale correcte. Cet article traite de la possibilité de l'implémentation sur un SoC (Système sur puce) embarqué d'une approche de suivi de caractéristiques basée sur un couplage serré des données inertielles et visuelles en utilisant l'algorithme KLT (Kanade-Lucas-Tomasi) [10]. Notre travail réside dans la réponse donnée aux contraintes des systèmes embarqués à forte intégration tels que les smart-phones [7] ou les micro-véhicules aériens [28]. Le SoC utilisé se compose d'une unité de traitement central CPU à architecture ARM et d'un processeur graphique GPU embarqué qui accélère le traitement. De plus, le coût élevé de calcul et la complexité de l'algorithme ont été considérés afin d'obtenir un système de suivi embarqué rapide et efficace.
Ce papier se divise en cinq sections: la section 2 présente les travaux actuels: le contexte, les applications et un aperçu de l'état de l'art des méthodes de détection, de description, de fusion et de suivi. La section 3 décrit l'algorithme KLT de suivi de caractéristiques assisté par les données inertielles ainsi que les principaux aspects de son implémentation sur un processeur graphique GPU. Ensuite, l'implémentation sur l'architecture embarquée Tegra est présentée dans la section 4. Dans la section 5, les résultats des expérimentations menées sont présentés et analysés, suivie par la conclusion et des propositions de travaux futurs dans la section 6.

2 État de l'art

De nombreuses applications de la vision par ordinateur ou de la robotique de haut niveau sont basées sur le suivi de caractéristiques particulières à travers plusieurs images; c'est par exemple le cas en vidéosurveillance, cartographie et navigation. Ce suivi consiste en une séquence d'opérations de recherche visant à extraire des caractéristiques visuelles sous forme de points d'intérêt (PoI) ou de régions d'intérêt (RoI) telles que des coins ou des zones texturées, et à les apparier sur les images entrantes. Selon les entrées considérées, trois approches peuvent être utilisées pour accomplir cette tâche [6][10][7] :
1) *algorithme de suivi basé vision :* il donne un suivi des caractéristiques précis et sans dérive, mais il est vulnérable aux mouvements rapides de caméra ainsi qu'à la navigation dans des environnements pauvres en texture et des arrière-plans complexes. De plus, il requiert un temps de traitement important.
2) *algorithme de suivi basé sur l'inertie :* cette approche peut traiter des vitesses très élevées et des mouvements accélérés [5], avec une réponse à haute fréquence. Toutefois, elle souffre d'une erreur de dérive accumulée dans le temps dans le cas d'une estimation à long terme.
3) *algorithme de suivi basé sur la fusion visuelle-inertielle :* grâce à la combinaison des avantages des deux technologies [6], le couplage des approches inertielle et visuelle permet d'améliorer la robustesse du suivi.

2.1 Principe des algorithmes de suivi

La chaîne de traitement globale diffère selon les données capteur utilisées (données visuelles ou/et inertielles). En général, la première étape des systèmes basés vision est la détection des PoI ou des RoI dans les images à l'aide d'un des détecteurs listés de manière non-exhaustive ci-dessous. L'étape suivante consiste à décrire ces PoI/RoI afin de construire une liste de caractéristiques, qui seront ensuite suivies d'une image à l'autre image. Dans ce travail l'objet suivi est représenté par des PoI puisqu'ils sont précis et génériques. Par la suite, nous nous intéressons aux algorithmes de détection et description de ces derniers.

Détection et sélection de caractéristiques. Les deux premières étapes des algorithmes de suivi basés sont au cœur de nombreuses recherches visant à accélérer l'exécution et à améliorer la qualité de la détection et de la description. Le détecteur de Harris [8] est l'un des algorithmes les plus utilisés [6][17]. FAST (*Features from Accelerated Segment Test*)[20] est un algorithme concurrent de Harris, il est efficace pour la détection des PoI mais il a une grande complexité calculatoire. Il existe également d'autres algorithmes qui combinent la détection et la description, tels que SIFT (*Scale Invariant Feature Transform*) [14], SURF (*Speeded-up Robust Feature*) partiellement inspiré de SIFT[3] ou ORB (*Oriented FAST and Rotated BRIEF*) [21]. Ce dernier est basé sur le détecteur FAST et le descripteur BRIEF (*Binary Robust Independent Elementary Features*) [21]. Il assure une détection rapide et une bonne performance algorithmique [7]. Une grande variété de méthodes de description des PoI a également été proposée, comme le descripteur DAISY [25], inspiré du descripteur SIFT. Il vise à réduire le temps de calcul et à mieux gérer les différents types d'invariance [25].

Algorithme de suivi des caractéristiques. Une fois les PoI détectés et décrits, le suivi des caractéristiques commence. L'un des algorithmes les plus utilisés est le KLT. Cette approche de suivi visuel est basée sur l'alignement des images [15], la détection et le suivi des caractéristiques [26], ainsi que la vérification et la sélection des bonnes caractéristiques à suivre [23]. En 2004, Baker, Simon et Matthews[2] ont proposé d'autres alternatives tel que l'ICIA (Inverse Compositional Image Alignment) moins complexes en calcul que KLT. Il consiste à diviser l'algorithme KLT en deux phases : le pré-calcul et l'itération (cf. section 3.1 et figure 2).
Dans ce travail, nous nous concentrons sur les algorithmes basés sur le couplage visuel-inertiel. Les étapes de l'algorithme de suivi visuel sont modifiées pour intégrer les données inertielles au cours d'une étape de fusion. Leur fonctionnalité dépend du type de couplage IMU/caméra utilisé (figure 1). Par conséquent, la méthode de fusion IMU/caméra influence la qualité et la complexité calculatoire de l'algorithme de suivi. Corey Montella [16] propose une étude des méthodes de fusion les plus populaires : filtre de Kalman (KF), Filtre de Kalman étendu (EKF) et filtre

particulaire (PF). Le PF présente une complexité calculatoire élevée par rapport aux autres filtres et ne convient pas aux systèmes embarqués [16]. Le KF est moins complexe à calculer que le PF mais ne peut être appliqué qu'aux systèmes linéaires. L'EKF et l'UKF (filtre de Kalman sans parfum) étendent l'application de KF aux systèmes non linéaires. Les deux méthodes présentent de bons résultats, l'UKF est légèrement plus efficace, mais plus complexe en calcul que l'EKF [1].

Dans notre contribution, l'algorithme développé dans [10], s'appuyant sur ICIA et intègrant les données de l'IMU, est implémenté sur une architecture embarquée basée sur GPU (Nvidia Tegra). La section 3 décrit cet algorithme et son implémentation.

2.2 Fusion des données IMU/caméra

La fusion IMU/caméra est utilisée pour diverses applications, telles que la réalité augmentée (AR) [18], la navigation [22][28] et les systèmes de suivi [10]. Cette dernière est le sujet principal de cet article. Cependant, la plupart des algorithmes existants ne sont pas optimisés pour les systèmes embarqués, notamment en termes de complexité calculatoire et de consommation d'énergie. Dans ce contexte, un nouvel éventail d'algorithmes spécialisés est développé. Par exemple, dans [7], un nouvel algorithme de suivi embarqué utilise des algorithmes réputés de détection (ORB), de fusion (EKF) et de suivi (KLT). D'autre part, un nouvel algorithme de rejet des *outliers* appelé LONSC (*LONgest Successive Consistency*) est également développé par [28]. De plus, un système hybride amélioré de suivi pour la réalité augmentée mobile (MAR) est proposé par [13]. Cette approche temps-réel associe un EKF à un suivi basé sur la détection de contours. Cependant, elle a encore besoin d'améliorations pour s'appliquer efficacement aux scénarios les plus réalistes.

Au vu de cet état de l'art,

nous constatons que le travail de M. Huangbo, J-S Kim et T. Kanade [9] est bien adapté à notre objectif. Il s'agit donc de profiter de la bonne qualité de suivi de cet algorithme en migrant vers un système temps réel embarqué avec une forte contrainte d'intégration. Dans les sections qui suivent l'algorithme traité dans [9] est présenté, suivi par son implémentation d'origine, puis de son implémentation sur des architectures embarquées NVIDIA Tegra ce qui représente notre contribution pour ce travail.

3 Suivi de points d'intérêt

Dans cette partie, deux approches conventionnelles de suivi sont présentées dans les sections 3.1 et 3.2 : l'algorithme de Kanade-Lucas-Tomasi (KLT) et son alternative ICIA (*Inverse Compositional Inverse Image*), en insistant sur les gains calculatoires de cette dernière solution. Les deux approches n'utilisent que des images et leur fonctionnement est intrinsèquement limité aux petits changements d'apparence entre images consécutives. Par conséquent, les mesures inertielles peuvent être introduites pour rendre le suivi plus robuste aux grands flux optiques, comme présenté dans la section 3.3. Enfin, l'implémentation de l'algorithme sur une architecture type PC (x86), comme dans [9], est passée en revue dans la section 3.4.

3.1 Algorithme KLT

Afin de suivre un point de l'image de l'instant t_1 à l'instant t_2, l'algorithme KLT vise à minimiser la somme de l'erreur quadratique entre deux images, l'image du modèle T qui est une sous-région extraite de l'image à l'instant t_1, et l'image I à t_2 reprojetée vers le repère des coordonnées du modèle [15]:

$$\sum_x [I(W(x;p)) - T(x)]^2 \tag{1}$$

où $x = (x, y)^\top$ sont les coordonnées de pixel, $W(.)$ est un modèle de suivi de mouvement et p est un vecteur de paramètres de déformation. Le modèle de suivi de mouvement utilisé est une transformation affine-photométrique [11]. Ainsi, le vecteur p a 8 paramètres $p = (a_1, ..., a_6, \alpha, \beta)$, où $(a_1, ..., a_6)$ sont liés à la déformation spatiale et (α, β) au changement d'éclairage. La différence entre deux images, présentée par δp, est la variation entre ces paramètres. L'expression à minimiser est ainsi :

$$\sum_x [I(W(x;p+\delta p)) - T(x)]^2 \tag{2}$$

L'utilisation de la méthode Gauss-Newton permet de déterminer δp :

$$\delta p = H^{-1} \sum_x [\nabla I \frac{\partial W}{\partial p}]^\top [T(x) - I(W(x;p))] \tag{3}$$

où H est la matrice *Hessienne* de taille $n \times n$:

$$H = \sum_x [\nabla I \frac{\partial W}{\partial p}]^\top [\nabla I \frac{\partial W}{\partial p}] \tag{4}$$

Enfin, la dernière étape consiste à mettre à jour les paramètres de déformation en utilisant le résultat de l'équation 3 :

$$p \leftarrow \delta p + p \tag{5}$$

Cette implémentation standard du KLT impose un coût calculatoire important égal à $O(n^2N + n^3)$ par itération, où le calcul de H prend $O(n^2)$ et son inverse $O(n^3)$, avec n le nombre de paramètres de gauchissement et N le nombre de pixels dans le modèle T [2]. Par conséquent, la mise à jour de H à chaque itération entraîne un coût important augmentant la complexité calculatoire de l'algorithme. Cette dernière peut être considérablement réduite si H est calculée une seule fois dans une étape de pré-calcul, comme expliqué dans la section suivante et illustré à la figure 2.

3.2 Algorithme ICIA

L'algorithme ICIA vise à préserver les mêmes performances algorithmiques que l'algorithme KLT tout en réduisant sa complexité calculatoire. Afin de ne pouvoir calculer H qu'une seule fois, et comme son nom l'indique,

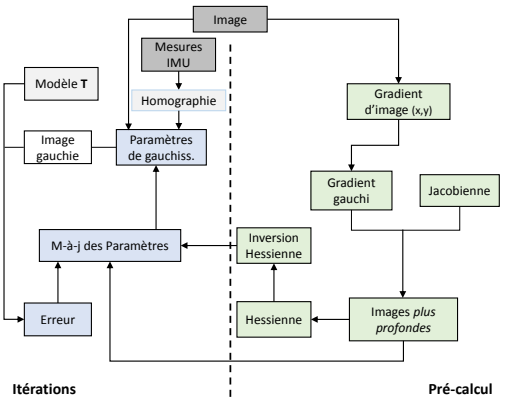

Figure 2: Algorithme de suivi de caractéristiques assisté par IMU (figure inspirée de [2]).

le rôle du modèle et de l'image est inversé pour aligner un modèle d'image $T(x)$ sur une image d'entrée $I(x)$:

$$\sum_x [T(W(x;\delta p)) - I(W(x,p))]^2 \quad (6)$$

δp est la mise à jour incrémentale des paramètres de déformation p. En conséquence, les matrices de gradient, jacobienne, hessienne et son inverse, sont pré-calculées. En implémentant ces étapes comme illustré dans la figure 2, la complexité passe de O($n^2N + n^3$) par itération à O($nN + n^3$) plus le coût de la phase de pré-calcul O(n^2), qui n'est consommé que lors de la mise à jour du modèle [2].

3.3 KLT assisté par des données inertielles

Afin d'améliorer les performances algorithmiques du suivi des points, Huangbo *et al.* [9] développent l'algorithme KLT assisté par les données inertielles en intégrant les mesures de l'IMU dans l'algorithme ICIA, comme l'illustre la figure 2. Cette information inertielle est introduite dans la formulation des paramètres de déformation afin de trouver un accord entre la complexité du modèle et la précision de suivi [12]. Cette méthode rend le suivi plus efficace et plus robuste, en exploitant la méthode de factorisation [27] et l'approximation du premier ordre pour calculer la valeur correcte de la vitesse angulaire. Ensuite, la matrice de rotation, représentée par des quaternions, est calculée. Enfin, l'homographie h est déterminée en utilisant les matrices de calibration K et de rotation R^t_{t+1} de la caméra, telle que :

$$h = K^{-1} R^t_{t+1} K \quad (7)$$

Les auteurs proposent deux implémentations de cet algorithme [10] : une version séquentielle sur CPU et une version accélérée utilisant un GPU de carte graphique) [12]. Elles sont réalisées sur des architectures de type PC de bureau non adaptées aux systèmes embarqués. Dans ce qui suit, nous reprenons l'implémentation initiale de cet algorithme (sous-section 3.4) avant de nous focaliser dans la section 4 sur l'adaptation pour le GPU embarqué Nvidia Tegra que nous proposons.

3.4 Implémentation sur architecture de bureau

Les GPU sont des architectures puissantes, peu coûteuses, flexibles et programmables [19]. Ils sont de plus en plus utilisés pour accélérer les parties complexes des applications temps-réel. Dans la littérature, de nombreuses implémentations d'algorithmes de suivi sur GPU existent [12][24][10]. Dans cette section, nous présentons brièvement l'implémentation de l'algorithme de suivi de [10] sur CPU et les choix effectués pour son accélération GPU.

Implémentation CPU. Pour l'implémentation de [10] sur le CPU Intel x86, la librairie OpenCV est utilisée pour les fonctions principales de traitement d'image (*p. ex.* alignement d'image). La librairie Intel *Integrated Performance Primitives* (IPP) est ensuite utilisée pour accélérer les opérations complexes d'OpenCV. De plus, toutes les étapes de suivi sont exécutées séquentiellement. Malheureusement, l'accélération de traitement atteinte ici ne suffit pas et il est encore possible de paralléliser certaines étapes du processus.

Accélération GPU. En se référant à la section 3.1, la complexité calculatoire de H est O(n^2) avec $n = 8$ en raison de l'utilisation du modèle affine-photométrique [9][10]. La complexité étant considérablement supérieure au modèle de translation où $n = 2$, une implémentation sur GPU s'avère cruciale pour limiter le temps d'exécution. Le suiveur de caractéristiques par KLT comporte deux fils de traitement : le fil de pixels et le fil de caractéristiques [12][10]. Dans le premier, les routines principales implémentées sont la pyramide d'images, le gradient d'image et la détection des PoI. Elles sont effectuées sur le pixel concerné et ses voisins [12][10]. Pour le second, les routines concernées sont la construction des tables de caractéristiques, le calcul des erreurs de déformation et la mise à jour des paramètres de mouvement. La figure 3 résume les fonctions principales de ces deux fils [12].

4 Implémentation embarquée sur l'architecture Tegra

Cette section détaille notre implémentation embarquée de l'algorithme KLT assisté par les données inertielles développée par [9]. Les architectures embarquées utilisées sont de type Tegra. Ce sont des SoC développés spécialement pour les appareils mobiles. Le NVIDIA Tegra K1 (2014), se caractérise par son GPU NVIDIA Kepler 192 cœurs et son CPU ARM Cortex-A15 64 bits en configuration 4+1. Cette combinaison de cœurs GPU et CPU permet d'atteindre des performances élevées et une efficacité énergétique importante. NVIDIA a proposé en 2015 un nouveau SoC plus avancé, le Tegra X1. Il se base sur le GPU NVIDIA Maxwell 256 cœurs et le CPU ARM Cortex-A57 64 bits associé au CPU ARM Cortex-A53 64 bits dans une

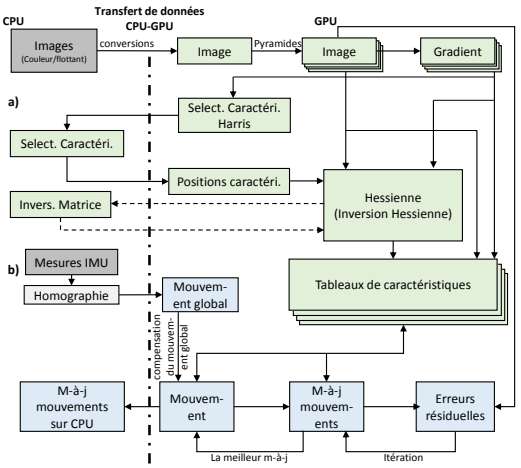

Figure 3: Distribution du processus sur GPU et CPU (version bureau): (a) fil de pixels, (b) fil de caractéristiques (figure inspirée de [12]).

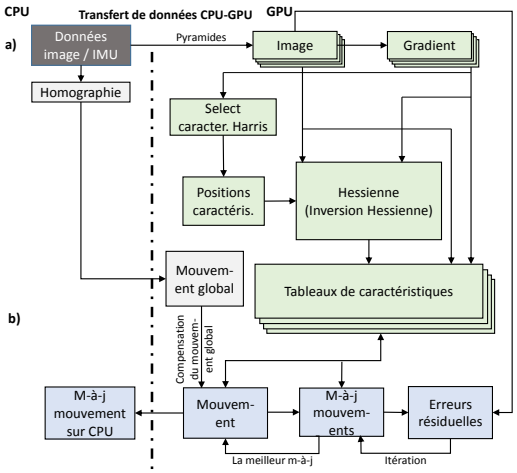

Figure 4: Distribution du processus (version embarquée): (a) fil de pixels, (b) fil de caractéristiques.

configuration 4+4. L'implémentation embarquée est illustrée dans la figure 4. Elle se décompose en trois étapes : le traitement des données (boîtes grises), la sélection des caractéristiques (boîtes vertes) et le suivi des caractéristiques (boîtes bleues), décrites ci-après.

4.1 Traitement des données

Les données des images et d'IMU (temporisations, biais, accélération et taux) sont pré-traitées pour être exploitées par la suite. La première étape comporte deux fonctions : le traitement d'images, où les conversions sont calculées et le filtre de lissage est appliqué, et le traitement des données IMU où l'homographie est calculée. Les mesures IMU sont échantillonnées à 100 Hz, tandis que les images sont générées à 30 Hz. Dans l'implémentation initiale sur l'architecture bureau, ces deux étapes sont exécutées de manière séquentielle, ce qui n'est pas compatible avec notre système en temps-réel et doivent être modifiées. En outre, elles consomment la plus grande partie du temps de calcul. Dans notre cas, l'image et les données IMU sont générées en temps réel et doivent être utilisées le plus rapidement possible après leur réception. Par conséquent, ces étapes sont exécutées en parallèle, en une seule étape, en choisissant le pire temps d'exécution des deux comme résultat final. Comme le montre le tableau 3, le traitement des données est une tâche très coûteuse en temps. Ceci est dû à la conversion d'image qui a un temps de calcul important sur le CPU hôte ARM en raison de la suppression de l'utilisation de la librairie IPP propre au processeur x86.

4.2 Sélection des caractéristiques

Une fois les données traitées, l'étape de sélection des caractéristiques exploite celles-ci pour détecter et décrire les PoI à l'aide du détecteur Harris et du modèle photométrique affine respectivement. Les sorties de cette étape sont utilisées pour construire le tableau des caractéristiques pendant l'étape de sélection, afin de les suivre dans l'étape de suivi.

4.3 Suivi des caractéristiques

Dans cette étape, la minimisation KLT est appliquée pour suivre les PoI détectés précédemment le long des images. De plus, les mises à jour des paramètres de mouvement sont traitées en utilisant les données inertielles.

5 Résultats

Dans cette section, nous commençons par décrire la méthodologie d'expérimentation. Puis, nous présentons les résultats du profilage, avant d'analyser les performances de notre implémentation. Nous utilisons un jeu de données d'images correspondant à une navigation dans une scène de bureau en intérieur, accompagnée d'un jeu de données IMU (temporisations, biais, accélération et taux). Ce jeu est donné par [9], les images (640x480, 30Hz) sont acquises par une caméra USB *Sentech* et l'ensemble de données IMU est généré par un capteur inertiel MEMS tri-axial O-Navi Gyroscube.

5.1 Cadre expérimental

L'algorithme est implémenté sur 4 architectures (tableau 1) : Intel Core2 Duo E8500 CPU (x86), l'implémentation de [10] (quad core Q9550 + GPU GTX280), NVIDIA Tegra X1 (TX1), et NVIDIA Tegra K1 (TK1). Cette sélection permet d'analyser et de comparer le temps de calcul de l'algorithme sur des architectures différentes (CPU de bureau, CPU-GPU de bureau et CPU-GPU embarqué) en spécifiant le temps d'exécution de chaque partie du code tel qu'il est présenté dans le tableau 3 et expliqué dans la section 4. Aussi, elle permet d'évaluer le nombre de FPS (image par seconde) et la précision (le rapport entre les PoI suivis avec succès et le nombre total des PoI) du suivi sur ces différentes architectures.

	Proc.	Cœur	Freq.	TDP
TK1:				8W.
CPU	32bits	4 A15+1	2.3Ghz	
GPU		192	0.8Ghz	
TX1:				10W.
CPU	64bits	4A57+4A53	1.9+1.3Ghz	
GPU		256	1Ghz	
[10]:				314W.
CPU: Q9550	64bits	4	2.83Ghz	95W.
GPU: GTX280			0.6Ghz	236W.
x86: E8500	64bits	2	3.16GHz	65W.

Table 1: Principales caractéristiques des architectures choisies ("TDP" pour "*Thermal Design Power*").

5.2 Apport des données inertielles

Tout d'abord, nous observons que l'exploitation des données inertielles dans le processus de suivi visuel a un effet primordial sur la précision (cf. tableau 2). Les pourcentages donnés se réfèrent à la précision expliquée ci-dessus. La qualité du suivi visuel diminue de façon significative lorsqu'il s'agit d'une inclinaison ou d'un mouvement panoramique, comme déjà démonstré dans [10]. En effet, le champ visuel peut facilement être perdu non seulement lorsque le mouvement est d'inclinaison ou panoramique, mais aussi lorsque le mouvement (rouleau, panoramique et inclinaison) est trop rapide. En outre, les PoI ne sont pas suivis avec succès. En introduisant des données inertielles, la qualité du suivi est bien meilleure (pas moins de 97%) quel que soit le type de mouvement et sa vitesse. Ces résultats, bien que déjà démontrés dans la littérature, confirment clairement l'intérêt de la stratégie de fusion visuelle-inertielle pour le jeu de données considéré dans ce travail.

Mouvement	Suivi visuel	Suivi inertiel-visuel
Panoramique	6%	97%
Inclinaison	2%	98%
Roulis	67%	99%

Table 2: Ratio de points d'intérêt suivis avec succès, selon le type de mouvement considéré.

5.3 Profilage sur différentes architectures

Pour assurer l'efficacité de notre contribution, nous établissons un profilage du code source sur différents dispositifs pour un même jeu de données. Le profilage est réalisé en instrumentant le code source du programme. Une temporisation est insérée au début et à la fin de chaque routine pour enregistrer ces temps de début et de fin. Grâce à ces informations, le profileur peut mesurer le temps réel que la routine prend pour chaque appel. Les résultats du profilage temporel par étape sont listés dans le tableau 3. Le temps total moyen par image est également indiqué.

L'étape la plus coûteuse, indépendamment de l'architecture choisie, est d'abord l'étape de **suivi des caractéristiques** à cause du calcul pyramidal [4], puis l'étape de **traitement des données**. Lorsque les composants Tegra (TX1 et TK1) sont utilisés, la durée du temps d'exécution de l'étape de traitement des données se justifie par la nécessité de convertir les données de entier à flottant (8u à 32f). Cette conversion est indispensable du fait de l'absence de la librairie IPP qui accélérait les fonctions OpenCV. Ainsi, les conversions prennent environ 70% du temps d'exécution du traitement des données.

Enfin, l'étape **sélection des caractéristiques** est la moins coûteuse en temps d'exécution total car elle n'est appelée que 34 fois en considérant 100 images de l'ensemble de données. Lorsqu'elle est exécutée, cette étape prend toutefois un temps d'exécution important. Les fonctions lourdes de cette étape sont le calcul des points de Harris et l'enregistrement des caractéristiques. En pratique, cette étape de mise à jour du modèle se produit seulement lorsqu'il manque des PoI pour le suivi.

Nous observons clairement que l'implémentation sur TX1 est plus rapide que celle sur TK1, elle tire principalement avantage de la vitesse/nombre de cœur du GPU et de l'architecture 64 bits (cf. tableau 1).

En plus du profilage temporel, nous présentons dans le tableau 3 les performances en termes de débit, exprimées en FPS traitées par le dispositif, et la précision du suivi associée, en plus de la puissance de dissipation thermique (TDP) pour chaque architecture (cf. tableau 1).

Nous observons que le coût du GPU dans l'implémentation bureau n'est pas négligeable. En comparant celle sur CPU-GPU de la littérature et notre implémentation sur x86, nous avons remarqué une baisse de 40% pour le FPS alors que le TDP diminue de 79%. La précision reste bonne dans les deux cas. Celle-ci reste élevée (environ 100 %) sur les architectures NVIDIA Tegra par rapport à l'implémentation sur l'architecture bureau. En ce qui concerne le FPS, il diminue de 71% alors que dans le même temps, le TDP diminue de 95% par rapport à [10]. Bien que ces résultats soient bons en termes d'efficacité énergétique, les performances en termes de FPS ne sont pas acceptables au regard du temps d'exécution sur les périphériques embarqués. La section suivante décrit les améliorations que nous avons considérées.

5.4 Améliorations algorithmiques apportées

L'implémentation embarquée offre un compromis intéressant entre la précision et le TDP. Elle doit cependant être encore améliorée pour réduire le temps d'exécution. La conversion (8u à 32f) est une étape chronophage ($\simeq$ 21% de traitement par trame) sur les NVIDIA Tegra. Les coûts de traitement se concentrent également sur l'étape de sélection. Nous avons donc conçu une amélioration algorithmique consistant à remplacer la version originale de Harris par une version améliorée. Dans notre implémentation ac-

	Étape	Sous-étape	[10]	x86
Architecture type PC	Suivi de caractéri.	Pyramides autres	10	40
	Sélect. de caractéri.	Harris Hessian	10*	10*
	Traitement de données	Conversion 8u-32f Conversion couleur/gris Homographie	9	-
	Total temps/image		40	60
	FPS		$\simeq 25$	15
	Précision %		$\simeq 99$	99.56
			TK1	**TX1**
Architecture Embarquée	Suivi de caractéri.	Pyramides others	160 12	60 5
	Sélect. de caractéri.	Harris Hessian	56* 45*	11.8* 8.9*
	Traitement de données	Conversion 8u-32f Conversion couleur/gris Homographie	60 13	36 7
	Total temps/image		340	138
	FPS		3	7.14
	Précision %		99.54	99.56

Table 3: Temps d'exécution des noyaux et temps total moyen par image traitée en "**ms**" sur les architectures de type PC et embarquée (* $\simeq$ 0ms quand non exécuté).

célérée de Harris (version *entière* de Harris), le nombre de PoI détectés est limité. En effet, l'image est décomposée en 8x8 zones, dans chacune le nombre de PoI à détecter est limité aux 16 meilleurs PoI de Harris (1024 PoI pour toute l'image). Cette méthode permet d'avoir un nombre limité de PoI et en même temps de bien les distribuer à travers l'image. Ainsi, dans le tableau 3, les résultats associés à la TX1 montrent le gain total à la fois dû aux améliorations algorithmiques sur Harris et à l'architecture.
Cette version accélérée de Harris sur GPU prend moins de 12ms par image sur TX1, alors que l'ancienne version implémentée sur TK1 prenait 56ms par image. En effet cette version accélérée utilise le type *entier* et permet à la fois la parallélisation au niveau pixel sur la première phase de détection et au niveau POI sur la deuxième phase sur les différents coeurs du GPU. Le temps d'exécution du calcul des points de Harris est ainsi réduit de 79 % et la conversion coûteuse de *entier* à *flottant* dans l'étape de traitement Harris est supprimée. Ces améliorations nous permettent de gagner globalement 52% de FPS, et de passer de 3 FPS sur TK1 à 7,14 FPS sur TX1.

6 Conclusion

Les étapes principales de traitement d'un système de suivi basé sur des capteurs visuels, inertiels et couplage inertiel/visuel ont été présentés dans cet article. Nous nous sommes focalisés sur l'algorithme de suivi KLT assisté par les données inertielles, qui a prouvé son efficacité en cas de mouvements complexes. Dans notre travail, nous proposons une implémentation de cet algorithme sur les GPU embarqués, TK1 et TX1, en adaptant certaines parties du code, notamment la modification du noyau CUDA du détecteur de Harris. Les expériences menées ont montré que nous gagnons en complexité de calcul et en temps d'exécution (32%) tout en maintenant une très bonne précision de suivi ($\simeq 100\%$). Cela prouve la faisabilité du portage sur dispositif embarqué d'un système de suivi robuste reposant sur KLT et des données inertielles.
Ces résultats ouvrent la voie vers de futurs améliorations qui se concentreront sur l'accélération du processus de suivi, en convertissant toutes les fonctions du code source au format *entier*, ou en explorant de nouvelles librairies , telles que "*VisionWorks*" et "*OpenVX*", afin de réduire le coût des fonctions de conversion. En particulier, trois fonctions critiques pourraient bénéficier de ces améliorations : le calcul des pyramides, l'alignement et l'enregistrement des caractéristiques, qui seront ciblées dans le but d'atteindre 30 FPS sur un dispositif embarqué.

References

[1] ARMESTO, L., TORNERO, J., AND VINCZE, M. Fast ego-motion estimation with multi-rate fusion of inertial and vision. *The International Journal of Robotics Research 26*, 6 (2007), 577–589.

[2] BAKER, S., AND MATTHEWS, I. Lucas-Kanade 20 years on: A unifying framework. *International journal of computer vision 56*, 3 (2004), 221–255.

[3] BAY, H., ESS, A., TUYTELAARS, T., AND VAN GOOL, L. Speeded-up robust features (SURF). *Computer vision and image understanding 110*, 3 (2008), 346–359.

[4] BOUGUET, J.-Y. Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm. *Intel Corporation 5*, 1-10 (2001), 4.

[5] CORKE, P., LOBO, J., AND DIAS, J. An introduction to inertial and visual sensing. *The International Journal of Robotics Research 26*, 6 (2007), 519–535.

[6] ELLOUMI, W., LATOUI, A., CANALS, R., CHETOUANI, A., AND TREUILLET, S. Indoor pedestrian localization with a smartphone: A comparison of inertial and vision-based methods. *IEEE Sensors Journal 16*, 13 (2016), 5376–5388.

[7] FANG, W., ZHENG, L., AND DENG, H. A motion tracking method by combining the IMU and camera in mobile devices. In *Sensing Technology (ICST), 2016 10th International Conference on* (2016), IEEE, pp. 1–6.

[8] HARRIS, C., AND STEPHENS, M. A combined corner and edge detector. In *Alvey vision conference* (1988), vol. 15, Citeseer, pp. 10–5244.

[9] HWANGBO, M., KIM, J.-S., AND KANADE, T. Inertial-aided KLT feature tracking for a moving camera. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2009), IEEE, pp. 1909–1916.

[10] HWANGBO, M., KIM, J.-S., AND KANADE, T. Gyro-aided feature tracking for a moving camera: fusion, auto-calibration and GPU implementation. *The International Journal of Robotics Research 30*, 14 (2011), 1755–1774.

[11] JIN, H., FAVARO, P., AND SOATTO, S. Real-time feature tracking and outlier rejection with changes in illumination. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on* (2001), vol. 1, IEEE, pp. 684–689.

[12] KIM, J.-S., HWANGBO, M., AND KANADE, T. Realtime affine-photometric KLT feature tracker on GPU in cuda framework. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on* (2009), IEEE, pp. 886–893.

[13] KUMAR, K., VARGHESE, A., REDDY, P. K., NARENDRA, N., SWAMY, P., CHANDRA, M. G., AND BALAMURALIDHAR, P. An improved tracking using IMU and vision fusion for mobile augmented reality applications. In *The International Journal of Multimedia and Its Applications (IJMA)* (2014), vol. 6, pp. 13–29.

[14] LOWE, D. G. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on* (1999), vol. 2, Ieee, pp. 1150–1157.

[15] LUCAS, B. D., AND KANADE, T. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2* (San Francisco, CA, USA, 1981), IJCAI'81, Morgan Kaufmann Publishers Inc., pp. 674–679.

[16] MONTELLA, C. The Kalman filter and related algorithms: A literature review. Tech. rep., Lehigh University, 2011.

[17] NIKOLIC, J., REHDER, J., BURRI, M., GOHL, P., LEUTENEGGER, S., FURGALE, P. T., AND SIEGWART, R. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on* (2014), IEEE, pp. 431–437.

[18] OSKIPER, T., SAMARASEKERA, S., AND KUMAR, R. Tightly-coupled robust vision aided inertial navigation algorithm for augmented reality using monocular camera and IMU. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on* (2011), IEEE, pp. 255–256.

[19] OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., HARRIS, M., KRÜGER, J., LEFOHN, A. E., AND PURCELL, T. J. A survey of general-purpose computation on graphics hardware. In *Computer graphics forum* (2007), vol. 26, Wiley Online Library, pp. 80–113.

[20] ROSTEN, E., AND DRUMMOND, T. Machine learning for high-speed corner detection. In *European conference on computer vision* (2006), Springer, pp. 430–443.

[21] RUBLEE, E., RABAUD, V., KONOLIGE, K., AND BRADSKI, G. Orb: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (2011), IEEE, pp. 2564–2571.

[22] SHEN, S., MICHAEL, N., AND KUMAR, V. Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft mavs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on* (2015), IEEE, pp. 5303–5310.

[23] SHI, J., ET AL. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on* (1994), IEEE, pp. 593–600.

[24] SINHA, S. N., FRAHM, J.-M., POLLEFEYS, M., AND GENC, Y. GPU-based video feature tracking and matching. In *EDGE, Workshop on Edge Computing Using New Commodity Architectures* (2006), vol. 278, p. 4321.

[25] TOLA, E., LEPETIT, V., AND FUA, P. A fast local descriptor for dense matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (2008), IEEE, pp. 1–8.

[26] TOMASI, C., AND KANADE, T. Detection and tracking of point features. *International Journal of Computer Vision* (1991).

[27] TOMASI, C., AND KANADE, T. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision 9*, 2 (1992), 137–154.

[28] ZHOU, G., YE, J., REN, W., WANG, T., AND LI, Z. On-board inertial-assisted visual odometer on an embedded system. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on* (2014), IEEE, pp. 2602–2608.