Méthode d'Optimisation Temporelle des Algorithmes de Vision

Patrick BONNIN[1,2], Laurent CABARET[1], Vincent HUGEL[1], Pierre BLAZEVIC[1]
Nacer M'SIRDI[1], Philippe COIFFET[1]

[1]Laboratoire de Robotique de Paris LRP
10-12 Av de l'Europe, 78 140 Vélizy

[2]Laboratoire de Transport et Traitement de l'Information L2TI et IUT de Villetaneuse, Dépt GEII
Av JB Clément, 93 430 Villetaneuse

{bonnin, cabaret, hugel, pierre, nacer, coiffet}@robot.uvsq.fr

Résumé – Compte tenu de la puissance de calcul disponible sur les robots mobiles, la recherche d'une optimisation temporelle devient un élément fondamental. Dans ce papier, une méthode est proposée, illustrée par deux applications réalisées. Enfin, nous présentons des résultats numériques, issus de « benchmarks » quantifiant les améliorations réalisées.

Abstract *– Taken into account the available computing power on mobile robots, the research of a temporal optimisation becomes a fundamental point. In this paper, a method is proposed, illustrated by two achieved applications. Then, numerical results, coming from benchmarks are presented, quantifying the realized improvements.*

1. Introduction

1.1 Puissance de Calcul et Vision Robotique

La puissance de calcul embarquée disponible aujourd'hui sur les robots mobiles de taille moyenne (de 50 cm à 1m) et même de petite taille (le robot AIBO de SONY mesure environ 25cm, voir [1] pour le détail du prototype) offre quasiment la possibilité d'effectuer des traitements à la cadence vidéo sur des images de faible résolution. Ceci permet le contrôle et la commande des robots par la vision, en temps réel, et en embarqué. Ainsi, les robots peuvent devenir totalement autonomes pour la réalisation de tâches encore relativement simples [2] [3]. Pour des tâches plus évoluées, un contrôle de haut niveau (supervision), effectué par un opérateur reste néanmoins possible [4].

1.2 Intérêt de l'Optimisation Temporelle

La recherche de l'optimisation temporelle des algorithmes de traitement d'image devient donc un élément fondamental. Permettant d'obtenir des gains en rapidité non négligeables (de l'ordre de la dizaine à la centaine), elle nous laisse envisager la possibilité d'implanter des algorithmes plus complexes, sur des images de résolution plus importante, sans avoir recours à des architectures spécifiques ou à des clusters de PC embarqués, ou même à devoir attendre pendant quelques années l'augmentation de puissance des processeurs classiques.

En un premier temps nous avons réalisé une optimisation temporelle de l'implantation des algorithmes constituant le Système de Vision du robot AIBO dans le cadre compétitif international de la RoboCup [5] (cf § 3). Puis nous cherché les concepts sous-jacents à cette application. Nous les avons formalisés (cf § 2), puis appliqués de nouveau dans un cadre plus général : la Vision Robotique. Cette dualité entre les applications et les concepts est fondamentale dans nos travaux.

1.3 Suite de l'Article

Dans la suite de l'article, nous commençons par présenter la méthodologie dérivée (§ 2.), que nous illustrons par deux applications : le système de vision temps réel embarqué du robot AIBO (§ 3.), et une chaîne algorithmique de détection des contours en imagerie couleur RVB ou YUV (§ 4.). Puis nous présenterons quelques quantifications des différentes optimisations (§ 5.), avant de conclure (§6.).

2. Méthodologie

2.1 Optimisation des Opérateurs de Bas Niveau

La méthodologie que nous proposons est basée sur la remarque suivante. Les traitements dits à bas niveaux, qui nécessitent le parcours de l'image sont très coûteux en temps de calcul. En effet, même une image de taille modeste par exemple 256 x 256 comporte 64K pixels, donc 64K octets en imagerie noir et blanc, 192K octets en imagerie couleur. La réalisation d'un quelconque opérateur nécessite au moins un parcours de l'image comportant à la fois le calcul de l'adresse permettant l'accès au pixel et le traitement local à effectuer. L'idée sous-jacente à l'optimisation est la suivante : il faut si possible diminuer le nombre de parcours de l'image, les rendre réguliers et optimiser le traitement local en chaque pixel.

Bien que se soit à bas niveau que l'on obtienne les optimisations les plus efficaces, il est néanmoins intéressant de continuer dans cette direction pour toute la chaîne de segmentation. En effet, compte tenu du peu de « marge » dont on dispose, le moindre gain est appréciable !

2.2 Méthodologie résumée sous forme de Règles

Ainsi, nous pouvons transcrire la méthodologie d'optimisation en 7 règles à respecter lors de l'implantation.

1. Le choix d'algorithmes simples, dont on peut implanter le parcours des données de manière régulière, sous la forme du balayage vidéo : ligne à ligne de haut en bas, et pour chaque ligne colonne par colonne de gauche à droite,
2. L'optimisation du calcul des adresses des pixels, en tenant compte que ceux-ci sont stockés selon l'ordre du balayage vidéo,
3. Le regroupement des traitements locaux de deux (ou plus) étapes successives constituant un même traitement. Un parcours d'image est « gagné » à chaque regroupement effectué. Mais pour cela, certaines contraintes doivent être respectées (cf § 4.1).
4. L'Optimisation des calculs locaux : éviter les calculs inutiles tels que les multiplications par +/-1 et 0, choisir l'utilisation de décalages à la place de multiplication / division par puissances de 2, effectuer les calculs en entier et non en réel (même si cela est moins pénalisant sur les nouveaux processeurs) etc...
5. Pour un même but, privilégier si c'est possible, l'emploi de filtres à niveau intermédiaire (sur les attributs des primitives) plutôt qu'à bas niveau (directement sur les pixels de l'image). En effet, les structures de données des primitives extraites sont plus rapides à parcourir que l'image,
6. Utiliser des structures de données à accès aléatoire (direct), plutôt qu'à accès séquentiel (balayage de la structure depuis le début). Ainsi, l'emploi d'un tableau de structures (il faut impérativement faire attention aux débordements) est plus rapide que celui d'une liste.
7. Dans le cas du traitement d'une séquence d'images, ce qui est le cas de la vision robotique, il faut gérer « statiquement » la mémoire, c'est-à-dire que les allocations dynamiques sont faites au départ (dans le constructeur en C++). Ceci préserve du danger de l'oublie d'une désallocation, et de la fragmentation de la mémoire au bout d'un certain temps ... n'oublions pas que 1800 images sont traitées par minutes à la cadence vidéo en NTSC (30Hz), 1500 en PAL, SECAM (25Hz).

2.3 Extension de l'Application de la Méthodologie.

Notons que ces règles ne s'appliquent pas uniquement à l'implantation logicielle de systèmes de vision : elles s'appliquent également à l'implantation matérielle sur composants électroniques ASICs, CPLDs, FPGAs. Nous les avons appliquées pour une implantation matérielle sur CPLDs, en visant un implantation future sur FPGAs [6]. En effet, la première règle de parcours des pixels de l'image selon le balayage vidéo garanti la possibilité du moins théorique de l'implantation matérielle sur de tels composants. La quatrième règle permet de réaliser un gain en blocs logiques, la troisième un gain en mémoires transitoires.

3. Le Système de Vision du Robot AIBO

3.1 L'Application

La tâche du système de vision est la détection, l'identification dans l'image puis la localisation sur le terrain de football des différents éléments de la scène : la balle, les buts, les joueurs, et les balises (pour se repérer sur le terrain), identifiables grâce à leur couleur ou combinaison de couleurs.

Le robot dispose d'un matériel de détection des couleurs, qui permet de classifier les pixels de l'image en 8 couleurs (au maximum) à la cadence vidéo, par multiseuillage dans l'espace YUV [1].

La première règle nous conduit à utiliser ce matériel en association avec un algorithme de décomposition en composantes connexes.

Au niveau des contraintes temporelles, compte tenu du contexte difficile de la robotique à pattes, où le déplacement d'un objet fixe dans la scène est chaotique dans la séquence d'images principalement au moment où la patte heurte le sol, il est impératif de se rapprocher le plus possible de la cadence vidéo (30 Hz, fréquence du NTSC).

Les benchmarks sont complexes à réaliser finement, car on ne peut pas dans une application réelle dissocier le module de vision des autres modules locomotion et stratégie, tous deux plus prioritaires, et tournant sur le même processeur.

3.2 Décomposition en Composantes Connexes

Parmi les diverses implantations possibles, notre choix s'est porté sur celle proposée par A. Rosenfeld, JL Pfalz [7], car le parcours des pixels de l'image suit le balayage vidéo. Deux balayages de l'image sont nécessaires. Le premier permet d'obtenir un étiquetage provisoire et de relever les équivalences entre étiquettes. Le second effectue la remise à jour l'image des étiquettes. Pour notre application, n'utilisant pas l'image des étiquettes, mais seulement les attributs des composantes connexes (centre de gravité, surface, boîte englobante), nous n'avons implanté que le premier.

Le « benchmark » que nous avons mis en œuvre est certes relativement grossier, mais il suffit amplement. Il englobe l'ensemble des algorithmes du module de vision : du bas niveau jusqu'au niveau intermédiaire assurant la fourniture au Module de Stratégie des informations pertinentes relevées dans l'image. Mais ces derniers traitements, agissant sur les structures de données, sont beaucoup plus rapides. Le benchmark est réalisé au niveau du module de Vision, mais au sein d'une application globale (l'attaquant) comportant les modules de Stratégie et de Locomotion, plus prioritaires [3]. Il consiste en une série d'appels au timer interne, sachant qu'un nombre important fixé a priori d'images (500 ou 1000) est traité entre deux appels successifs. La cadence obtenue est de 30 images par seconde, soit la fréquence en NTSC.

Cette simple implantation a été guidée par les règles :

- n° 1 : le choix d'un algorithme simple utilisant le parcours des données sous forme de balayage vidéo,
- n° 2 : nous avons optimisé le calcul des adresses, une

simple incrémentation de pointeurs remplace son calcul

- n° 6 : les données sont organisées en tableaux de structures,
- n° 7 : Toutes les structures de données : images et tableaux de structures sont crées dynamiquement et initialisées une fois pour toutes dans le constructeur de la classe Vision.

Au niveau des résultats, l'éclairage bien trop puissant des terrains de compétition génère des composantes connexes « artéfacts » qu'il faut impérativement éliminer ou du moins atténuer par un filtrage à bas niveau, car nous n'avons pas réussi par un filtrage à niveau intermédiaire (sur attributs des composantes connexes) comme le préconise la règle 5.

3.3 Filtrage de Bas Niveau

Plusieurs filtrages de type « morphologiques » sur des images de multiples niveaux (9 au maximum dans notre cas) ont permis d'obtenir des résultats satisfaisants : le filtrage majoritaire, l'ouverture, les filtrages de type bord et coin (cf figure n° 1).

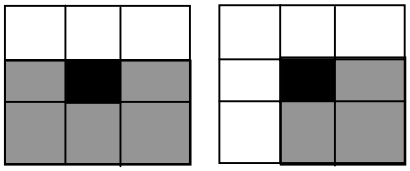

FIG. 1 : filtrages Bord et Coin

Au niveau temporel, le filtrage majoritaire est de loin le plus long. L'ouverture nécessite deux balayages vidéo de l'image, les filtres bord et coin un seul.

La combinaison : Filtrage de Bas Niveau + Décomposition en Composantes Connexes + Traitements de niveau intermédiaire permet d'obtenir une cadence d'environ 18 à 20 Hz en utilisant les filtrages de bas niveau les plus rapides, c'est-à-dire de type bord et coin. Selon notre benchmark, ils sont tous les deux temporellement équivalents.

Selon la troisième règle évoquée, l'intégration au sein d'un seul et même balayage vidéo de l'image des tests de connexité (pour la décomposition en composantes connexes) aux tests de configuration du voisinage (filtrages de type bord ou coin) est possible. Elle permet d'obtenir les mêmes résultats, tout en supprimant un balayage vidéo, ce qui assure selon notre benchmark une cadence de traitement de 30 Hz.

4. Détection des Contours

4.1 Exemple d'une chaîne d'Opérateurs à Regrouper et Règle de Regroupement

Pour un algorithme plus complexe, telle que une détection des contours, il faut optimiser un nombre plus important d'opérateurs séquencés : par exemple une détection de gradients (en norme et argument), un seuillage sur la norme du gradient, un affinage, une prolongation puis un chaînage des contours et une approximation polygonale.

Il est possible de décomposer le voisinage 3x3 centré du pixel courant, selon le parcours vidéo de l'image en 3 voisinages : - présent (constitué du pixel examiné), - passé (constitué des pixels ayant déjà été examinés : sur la ligne du dessus, et à gauche), - futur (constitué des pixels restant à examiner : à droite et sur la ligne du dessous).

La règle de regroupement est simple :

Un traitement local peut être regroupé aux traitements locaux précédents que s'il nécessite uniquement (au maximum) les résultats des traitements précédents sur les voisinages passé et présent.

En effet, lorsque l'on effectue le traitement sur le pixel courant, l'information des traitements précédents n'est disponible que sur les voisinages passé et présent. Elle n'est pas disponible sur les pixels du voisinage futur qui n'ont pas encore été traités.

En modifiant l'algorithme d'Affinage des Contours (cf § 4.2), nous allons le regrouper aux traitements précédents. En revanche, il est impossible de regrouper l'algorithme de prolongation des contours. Ainsi la chaîne d'opérateurs précédemment évoquées peut s'implanter en deux regroupements (c'est-à-dire en utilisant deux balayages vidéos). Le premier constitué du calcul des gradients, du seuillage sur la norme et de l'affinage. Le second est constitué de l'étape de prolongation, et de l'étape de chaînage des contours, toutes les deux imbriquées. Pour plus de détail, se reporter à [8].

4.2 Algorithme d'Affinage

Par définition, un point est de contour si et seulement si la norme de son gradient est maximale locale (dans son voisinage 3 x 3 centré) directionnelle dans la direction du gradient.

L'inconvénient de cette définition, est qu'il faut comparer la norme du gradient du pixel courant avec la norme du gradient de deux pixels voisins dont l'un est dans le futur.

Moyennant l'hypothèse de la cohérence de l'argument du gradient (vérifiée expérimentalement), la comparaison avec le pixel dans le passé est suffisante. Ceci permet l'application de la règle de regroupement.

5. Exemples de « Benchmarks »

5.1 Conditions Expérimentales

Il est nécessaire de tenter de quantifier les différents gains des règles énoncées au § 2, pour - d'une part s'assurer que les concepts sous-jacents sont fondés, et - d'autre part pour évaluer si une nouvelle implantation est justifiée.

Les résultats dépendent du type de processeur. Aussi, nous précisons qu'il s 'agit ici d'un Pentium III, cadencé à 500Mhz, tournant sous le noyau 2.2 de Linux (distribution Mandrake 7.2). Les mesures sont effectuées à l'aide de la fonction système clock(), qui inclus le temps du système

d'exploitation, et dont la précision est 1/100ème s. Pour améliorer la précision de nos mesures, nous effectuons 100 itérations du programme à évaluer. Une statistique sur une dizaine d'échantillons nous a montré la stabilité des mesures.

Contrairement au § 3, nous allons évaluer « finement » les optimisations temporelles du système de vision.

5.2 Optimisation de l'Adressage et des Calculs

L'algorithme de benchmark choisi est le filtre moyen, car il est suffisamment représentatif pour les calculs et l'accès aux données. Le traitement de chaque pixel requiert 9 lectures, 1 écriture, 9 additions et une division.

Pour quantifier l'optimisation relative à l'adressage nous allons comparer 3 implantations :

- Avec « encapsulation » des données : la lecture et l'écriture des données sont effectuées à l'aide de méthodes. Cette technique, la plus longue est prise en référence.
- Avec accès aléatoire : l'adresse du pixel est calculée de manière absolue,
- Avec accès séquentiel : l'adresse du pixel est obtenue à partir de l'adresse du pixel précédent, par simple incrémentation.

Les formules sont pour le gain et l'accélération :

G = (Tréf – T)/ Tréf et A = T/Tréf

Implantation	Temps	Gain(%)	Accélération
« Encapsulation »	0.0848	0	1
Acc. Aléatoire	0.0609	28	1.4
Acc. Séquentiel	0.0117	86	7.24

Pour quantifier l'optimisation des calculs nous allons comparer deux implantations :

- Avec l'opérateur de convolution (9 « * » et 9 « + » et 1 « / »), par pixel
- Simplement : 9 « + » et 1 « / » par pixel

Implantation	Temps	Gain(%)	Accélération
Convolution	0.0714	0	1
Additions	0.0609	14	1.17

5.3 Optimisation du Regroupment des Calculs Locaux

L'algorithme de benchmark choisi est le filtre de Kirsh à 4 directions, pour le nombre de regroupements possibles. Ainsi pour quantifier l'intérêt de ces regroupement nous allons tester 3 implantations différentes :

- Toutes les étapes sont séparées : le filtre est composé de 4 convolutions et d'une étape de synthèse des résultats,
- Le regroupement de ces étapes, avec un accès aléatoire,
- Idem avec un accès séquentiel.

Implantation	Temps	Gain (%)	Accélération
Séparation	0.3060	0	1
Acc. Aléatoire	0.1192	61	2.56
Acc. Séquentiel	0.0300	90	10.2

5.4 Conclusion

Sur deux exemples simples et représentatifs, nous remarquons qu'il est aisé d'obtenir une accélération non négligeable, d'un facteur 5 à 10, ce qui est appréciable lorsque l'on manque de puissance de calcul embarquée !

6. Conclusion

Compte tenu des puissances de calcul embarquées disponibles aujourd'hui sur les robots mobiles, la recherche de l'optimisation temporelle des algorithmes de vision est un élément fondamental pour permettre aux robots d'être autonomes, pour la réalisation de tâches encore relativement simples. Nous avons proposé une méthodologie, traduite sous forme de règles, que nous avons illustré par deux applications concrètes réalisées. Enfin, nous avons présenté des résultats issus de « benchmarks » simples permettant de quantifier les améliorations obtenues.

7. Références

[1] M. Fujita, S. Zrehen, H.Kitano, *A Quadruped Robot for RoboCup Legged Robot Challenge*, Proceedings of the second RoboCup Workshop, Paris Juillet 98

[2] V.Hugel, P.Bonnin, P.Blazevic, *Using reactive and adaptive behaviors to play soccer*, AI magazine, vol 21, n°3, Fall 2001 pp 53-59

[3] Vincent Hugel, Partick Bonnin, Ludovic Raulet, Pierre Blazevic, *Vision based behavior strategies to play soccer with legged robots*, RoboCup99: Robot Soccer World Cup III, éditions Springer Verlag, M. Veloso, E.Pagello, H.Kitano Eds.

[4] M.Chetto et al, *Projet « Cléopatre »*, RNTL démarrage Septembre 2001

[5] M. Veloso, E.Pagello, H.Kitano, *RoboCup99: Robot Soccer World Cup III*, éditions Springer Verlag.

[6] P.Bonnin, L. Cabaret, V.Hugel, P.Blazevic, N.M'Sirdi, P.Coiffet, *Exemple de Conception et de Réalisation d'un Système de Vision pour la Robotique Mobile et Autonome*, Int. Conf on Image and Signal Processing ICISP 2001, Agadir Maroc 3-5 Mai 2001.

[7] A. Rosenfeld, JL Pfalz, *Sequential operations in digital picture processing*, Journal of ACM, vol 13, n°4 1966

[8] Patrick Bonnin, *Vers une méthode de Conception et de Réalisation de Systèmes de Vision Temps Réel Embarqués pour la Robotique Mobile et Autonome*, HDR soutenue le 25/01/2000 à l'Univ de Versailles.