Real-time multi-target tracking : a study on color-texture covariance matrices and descriptor/operator switching

Andrés Romero Mier y Teran

UNIVERSITÉ PARIS-SUD

ÉCOLE DOCTORALE : Sciences et Technologies de l'Information, des Télécommunications et des Systèmes (STITS)

Laboratoire de Recherche en Informatique (LRI)

Équipe : ParSys

DISCIPLINE : Informatique et traitement du signal

THÈSE DE DOCTORAT

par

ANDRÉS ROMERO MIER Y TERÁN

REAL-TIME MULTI-TARGET TRACKING: A STUDY ON COLOR-TEXTURE COVARIANCE MATRICES AND DESCRIPTOR/OPERATOR SWITCHING.

Directeur de thèse :	Lionel Lacassagne	Maître de conférences, Université Paris-Sud (LRI)
Co-encadrante	Michèle Gouiffès	Maître de conférences, Université Paris-Sud (LIMSI)

Composition du jury :

Rapporteurs :	Guy Gogniat	Professeur, Université de Bretagne-Sud, France
	Alice Caplier	Professeur, Grenoble-INP, France
Examinateurs :	Alain Tremeau	Professeur, Université Jean Monnet, Saint Etienne, France
	Thierry Chateau	Professeur, Université Blaise Pascal, Clermont-Ferrand, France
Invité spécial :	Gregory Ligny	Consultant Senior chez Thales University Consulting

Contents

Suivi temps-réel : matrices de covariance couleur-texture et commutation automatique de descripteur/opérateur

Synthèse du manuscrit en version française

0.1 Introduction

Cette thèse propose un système de vision par ordinateur capable de détecter et suivre plusieurs objets dans les séquences vidéo. L'algorithme proposé de recherche de correspondances s'appuie sur les matrices de covariance obtenues à partir d'un ensemble de caractéristiques extraites des images (couleur et texture principalement). L'idée initiale sur laquelle reposent les bases de notre algorithme a été publiée par Porikli *et al*. [109] et son principal avantage est l'utilisation d'un descripteur de l'objet à suivre fusionnant des sources d'information très hétérogènes. Cette représentation, compacte et discriminante est efficace pour le suivi d'objets mais également leur ré-identification.

Quatre contributions sont introduites dans cette thèse. Tout d'abord nous nous intéressons à l'invariance des algorithmes de suivi face aux changements de contexte. Nous proposons ici une méthodologie pour mesurer l'importance de l'information couleur en fonction de ses niveaux d'illumination et de saturation. Ensuite, une deuxième partie se consacre à l'étude des différentes méthodes de suivi, leurs avantages et limitations en fonction du type d'objet à suivre (rigide ou non rigide par exemple) et du contexte (caméra statique ou mobile). La méthode que nous proposons s'adapte automatiquement et utilise un mécanisme de commutation entre différentes méthodes de suivi en considérant leurs forces complémentaires [79]. Notre algorithme s'appuie sur un modèle de covariance qui fusionne les informations couleur-texture et le flot optique (KLT) modifié pour le rendre plus robuste face aux changements d'illumination [113]. Une deuxième approche propose l'analyse des différents espaces de réprésentation couleur afin d'obtenir un descripteur assurant un bon équilibre entre pouvoir discriminant et invariance photométrique. Nous avons proposé une nouvelle représentation par matrice de covariance qui par une nouvelle façon d'intégrer les caractéristiques de texture LBP (pour *Local Binary Patterns* en anglais) permet de réduire la taille des matrices tout en conservant un très bon pouvoir discriminant [114].

Une troisième contribution porte sur le problème de suivi multi-cibles où plusieurs difficultés apparaissent parmi lesquelles la confusion des identités de deux objets, les occultations, la fusion ou la division des trajectoires. Ici, un ensemble de matrices de covariance distribuées spatialement est utilisé pour ré-identifier les cibles. La solution finale que nous proposons utilise une fonction d'énergie discrète qui s'appuie sur le comportement de l'ensemble des trajectoires et sur les modèles d'apparence proposés [116]. Finalement, à partir des algorithmes développés, nous avons réalisé une étude

sur l'adéquation algorithme-architecture et implémenté les codes sur des architectures multi-cœur en atteignant des gains importants en terme de rapidité d'exécution [79, 117].

Ainsi la thèse se divise en six parties. Tout d'abord, nous introduisons dans la partie 0.2 les caractéristiques visuelles disponibles au niveau pixel et leur utilisation pour définir le descripteur plus global de l'objet à étudier. Nous abordons ensuite dans 0.3 le problème de la détection d'objet et du suivi. La troisième partie 0.4 détaille les premières contributions de la thèse : la commutation d'opérateurs et de caractéristiques pour le suivi. Ensuite, nos approches proposées en ce qui concerne la mise en correspondance par covariance font l'objet de la partie 0.5. Enfin, l'étude algorithme-architecture est développée dans la partie 0.6.

0.2 Information locale et représentation d'objets

La qualité d'un descripteur associé à un objet peut être mesurée suivant trois critères :

- la compacité de la représentation, pour permettre un stockage et une exécution efficaces ;
- le caractère distinctif ou discriminant, c'est-à-dire que chaque descripteur doit être unique pour chaque objet de la vidéo considérée ;
- la répétabilité : la représentation doit être constante au cours du temps malgré les modifications des conditions d'acquisition ;

À partir des multiples informations pixelliques disponibles (voir paragraphe 0.2.1), plusieurs descripteurs d'objets sont envisageables (paragraphe 0.2.2).

0.2.1 Information disponible au niveau pixel

La couleur, la luminance. La vision humaine est fondamentalement tri-chromique : elle est basée sur les réponses de trois types différents de cônes de photo-récepteurs situés dans notre rétine (LMS). Pour des raisons historiques, la plupart des caméras fournissent des informations de couleur dans un système RGB mais il existe bien d'autres espaces de représentation (par exemple XYZ, les systèmes de couleurs opposées, $L^*u^*v^*$, $L^*a^*b^*$, HSV, HSI, et HSL) [136, 21]. Remarquons également les modèles de couleurs gaussiens qui correspondent à une nouvelle théorie de la mesure de la couleur basés sur la théorie espace-échelle dans le domaine spatio-spectrale [51].

Contours et gradients. Les dérivées de l'image sont des données essentielles pour décrire la structure locale des images et leurs applications en vision par ordinateur sont vastes : détection des contours, extraction de caractéristiques, flux optique, la segmentation d'image et la détection d'objet. Les opérateurs de gradient les plus populaires sont ceux de Sobel [105] et de Canny [22] et il existe plusieurs approches dédiées à la couleur : l'approche vectorielle de Di Zenzo [35], l'approche de Carron [24] qui fusionne les gradients de teinte, saturation et luminance en accordant une importance plus grande aux couleurs de forte saturation.

Texture. Classiquement, les bancs de filtres tels que celui de Gabor [42] sont efficaces mais difficiles à utiliser dans des applications temps-réel. Récemment des méthodes plus économiques et tout aussi discriminantes telles que les Motifs Binaires Locaux LBP (pour *Local Binary Patterns*) [101] ont vu le jour. L'opérateur consiste à analyser un voisinage

circulaire autour d'un pixel central. La valeur du pixel central est utilisée comme un seuil pour ses voisins. Si un voisin est de valeur supérieure, alors le code qui lui sera associé dans le LBP est 1, et sinon 0. Enfin, chaque bit dispose d'un poids en fonction de sa position dans le voisinage, et l'on peut en déduire une valeur décimale caractéristique de ce motif local. Il s'agit donc d'une représentation à la fois compacte et peu sensible aux changements de contraste. Après une simple normalisation, ils deviennent invariants aux changements de rotation.

Primitives locales et points d'intérêt. Il existe un nombre considérable de méthodes de détection et de description de points d'intérêt : Harris [124], SIFT [89], SURF [10] et FAST [119].

Information de mouvement. Dans le cas d'une caméra fixe, les informations de mouvement peuvent être extraites aisément pas soustraction de fond. Dans un cadre plus général, et en particulier lorsque la caméra est mobile, on s'intéressera plutôt à des approches de calcul de flot optique [66, 90].

Information de profondeur. Enfin, avec l'émergence des capteurs RGB+D de type Kinect, il est désormais devenu aisé d'obtenir des cartes de profondeurs. Les méthodes alternatives, basées sur la stéréo-vision et la lumière structurée offrent par ailleurs des possibilités étendues dans certaines conditions d'acquisition, pour des distances capteur-objet supérieures à 5 mètres ou encore en extérieur.

0.2.2 Représentation d'un objet

À partir des informations disponibles au niveau pixel, différentes représentations plus globales sont possibles. L'objet peut être directement représenté par le bloc de pixels qui lui est associé dans l'image, où chaque pixel porte une information de luminance, de couleur (dans différents espaces de représentation) ou de profondeur par exemple. Ensuite, il peut être décrit par un histogramme, de couleur [54] ou de gradients orientés HOG [34] ou par son contour [76] ou bien encore par une matrice de covariance de caractéristiques pixelliques [139]. Comparée aux représentations par histogrammes, ces derniers descripteurs sont de dimension plus faible et leur taille ne dépend pas de la taille initiale de l'objet dans l'image. De plus, ils peuvent être calculés rapidement par l'usage d'images intégrales. C'est sur cette représentation que s'appuie une grande partie de nos travaux de thèse.

0.3 Détection et mise en correspondance

La reconnaissance visuelle est un problème étroitement lié à l'apprentissage des catégories visuelles à partir d'un ensemble limité d'instances. Typiquement deux approches sont utilisées pour résoudre ce problème: l'apprentissage des catégories génériques et la ré-identification d'instances d'un objet particulier. Dans le dernier cas, il s'agit de reconnaître un objet en particulier: le portrait sur un magazine, une liste de personnes enregistrées sur une base de données, des monuments connus tel que la tour Eiffel. D'autre part la reconnaissance générique consiste à retrouver toutes les instances d'objets qui appartiennent à la même catégorie conceptuelle : toutes les voitures, les piétons, les oiseaux, etc. Pour la reconnaissance spécifique, les algorithmes visent à trouver des correspondances ou à faire du *matching* en utilisant des descripteurs propres à l'instance observée. La reconnaissance générique utilise des modèles statistiques de l'apparence et la forme des objets, afin de chercher dans de nouvelles images inconnues les zones de forte similarité avec le modèle statistique. Ce type d'apprentissage requiert l'accumulation d'un ensemble d'images d'entraînement pour extraire ou construire le mod-

èle de chaque catégorie. Trouver les correspondances existantes entre deux (ou plusieurs) images n'est pas un problème aisé. Un objet peux générer deux instances complètement différentes en fonction des situations non contrôlées telles que l'illumination, le point de vue de la camera, la position de l'objet, les occultations et l'encombrement.

Les approches de classification les plus répandues dans la communauté de la vision par ordinateur sont les approches SVM (pour *Support Vector Machine* en anglais) et Adaboost. Dans le cadre de notre étude concernant l'utilisation des matrices de covariance, nous nous intéressons également à la classification dans les espaces Riemaniens. Des descriptions plus complètes aux SVM et à leur implémentation peut être trouvée dans [88] et [17]. Dans le domaine de la vision par ordinateur, on se réfère très souvent aux travaux de [34] sur la classification piéton/non piéton par HOG (histogrammes de gradients orientés) associés aux SVM. Concernant Adaboost, les travaux précurseurs concernent la détection de visages en utilisant des ondelettes de Haar [144].

Enfin, la classification piéton/non piéton a également été effectué dans un espace Riemanien [140] en utilisant une représentation par matrices de covariance. L'approche est inspirée par l'algorithme *LogitBoost* [48] qui opère sur les espaces vectoriels.

Concernant les métriques utilisées, de nombreuses méthodes ont été proposées par la communauté de la vision par odinateur. Quatre approches ont particulièrement inspiré nos travaux.

Appariement de blocs ou ***Template matching*****.** Il s'agit de comparer directement les images associées aux objets en utilisant de simples mesures, telles que l'erreur quadratique moyenne *MSE* (pour *Mean Squared Error*) , la corrélation normalisée croisés NCC (pour *Normalized Cross-Correlation*) ou encore l'indice de similarité de structure $SSIM$ (pour *Structural SIMilarity index*). Ces approches s'avèrent plus adaptées pour mettre en correspondance ou suivre temporellement des zones de petite taille et des objets planaires dont le mouvement est rigide.

Le flot optique médian plus populairement connu sous le nom de *Flock of Trackers* FoT est une approche capable de suivre des objets non-rigides par analyse du flot optique obtenu par l'approche de suivi KLT (Kanade-Lucas-Tommasi) en utilisant l'opérateur médian pour estimer la direction de déplacement de l'objet. Quand une cible est détectée, son rectangle englobant est partitionné en quadrants. Chacun d'entre eux est suivi par KLT, les 50% de points les plus erronés sont écartés, et le reste participe au calcul du déplacement global. Cette méthode apporte une certaine robustesse par rapport aux occultations partielles. Elle est adaptée dans le cas d'objets rigides montrant des points saillants.

Appariement d'histogrammes et ***Mean-Shift*****.** Dans le cas du suivi *Mean-Shift* [30], l'objet est modélisé par une représentation couleur-espace, et le suivi est fait par descente de gradient utilisant la distance de Bhattacharyya. L'histogramme est généralement quantifié afin d'assurer une exécution temps-réel et d'éviter les représentations creuses. Cette approche est particulièrement adaptée aux objets dont les couleurs sont caractéristiques par rapport à celles du reste de l'image. Elle permet une bonne résilience vis-à-vis des déformations de l'objet.

Le suivi par covariance. L'objet à suivre est représenté par sa boîte englobante et en chaque point (x, y) de l'objet est calculé un vecteur de caractéristiques $F(x, y)$. La combinaison originalement proposée par Tuzel *et al*. [139] continent la position des points (x, y), leur couleur (composantes RGB) et la norme des 1ère et 2nde dérivées d'intensité par rapport à x et y. Ainsi, chaque point est converti en un vecteur de dimension 9 :

$$F(x,y) = \left[\begin{array}{ccccccccc} x & y & R(x,y) & G(x,y) & B(x,y) & \left|\frac{\partial I(x,y)}{\partial x}\right| & \left|\frac{\partial I(x,y)}{\partial y}\right| & \left|\frac{\partial^2 I(x,y)}{\partial x^2}\right| & \left|\frac{\partial^2 I(x,y)}{\partial y^2}\right| \end{array} \right]^T,$$

L'objet est ensuite représenté par la matrice de covariance de taille 9×9 obtenue à partir de l'ensemble de ces vecteurs. Le suivi consiste alors à retrouver dans la nouvelle image la région dont la matrice de covariance est la plus similaire. Cela peut être fait par une recherche exhaustive [139] à la fois sur la position et sur l'échelle. Cette approche est performante pour suivre tout type d'objet, particulièrement lorsqu'il n'y a aucune information disponible sur la dynamique de l'objet ou que celui subit des mouvements erratiques entre deux images. Plusieurs approches ont également été proposées pour incorporer des données concernant la dynamique de l'objet et estimer le déplacement le plus probable.

0.4 Adaptation au contexte et commutation de méthode

Aucune des méthodes de suivi évoquées dans la partie 0.3 n'est applicable de manière universelle, chaque algorithme ayant ses avantages et ses limites. Les performances qu'ils offrent sont bien souvent dépendantes de l'application et il n'existe pas d'algorithme qui puisse être considéré comme *supérieur* aux autres. Quelques-uns des aspects à prendre en compte dans la définition d'un algorithme de suivi sont les suivants : la nature de la cible, les changements de conditions d'éclairage, les variations de pose ou d'apparence, les déformations non -rigides et des occlusions totales ou partielles. Malheureusement, dans de nombreuses applications telles que la vidéo-surveillance, il n'y a aucune information préalable disponible sur la nature de la cible (rigide ou non-rigide) et l'estimation de ces informations à la volée peut poser un problème très difficile, c'est particulièrement vrai lorsque les objets sont détectés en utilisant des algorithmes de soustraction de fond au lieu d'un classificateur d'objets. Cette thèse soutient qu'une *bonne* méthode de suivi doit résulter de la combinaison de plusieurs approches, l'algorithme qui en résulte devant être capable de gérer une plus grande liste de difficultés que tous les algorithmes originaux tout en étant suffisamment rapide. L'objectif est d'obtenir un bon équilibre entre la robustesse et la précision tout en conservant des besoins en puissance de calcul aussi faibles que possible afin de maintenir l'exécution en temps réel. Ici trois approches sont proposées.

0.4.1 Adaptation aux changements de saturation

Principe. Au cours d'une séquence d'images il est possible de rencontrer de fortes variations de luminosité, qui ne sont pas toujours uniformes dans l'image. Il n'est donc pas aisé de définir au préalable les meilleures caractéristiques à utiliser : couleur ou luminance.

Le traitement couleur est généralement plus coûteux en temps de calcul tandis que la luminance est par essence sensible aux variations photométriques. La couleur permet de définir de précieux invariants couleur [54] qui sont malheureusement peu fiables dans le cas de faibles saturations. Nous proposons une approche de suivi qui sélectionne automatiquement les caractéristiques en fonction de leur pertinence : la luminance sera utilisée aux pixels pour lesquels la saturation de la couleur est faible et la couleur sera utilisée dans les autres cas. Cette approche a été validée dans le cadre du suivi de points de type *KLT* [90, 133]. La composante couleur utilisée a été choisie de manière à être compacte et invariante aux changements d'illumination.

Choix de la composante couleur. Une façon directe de séparer l'information de chrominance (invariante) de celle de luminance est de normaliser (norme L_1) chaque composante (R, G, B) en les divisant par la somme des trois, qui correspond à l'intensité de la couleur. On obtient les composantes (r, g, b) qui ne dépendent plus que de l'*albedo*, propriété

intrinsèque de réflectance du matériau.

À partir de ces trois composantes, nous calculons la valeur scalaire $L_1 = max(r, g, b)$ qui représente la composante la plus saturée parmi les trois. Notons que cette information de couleur est plus facile à manipuler que les trois composantes et conserve l'invariance. Elle s'avère toutefois moins discriminante. Dans le cas du suivi KLT où la cohérence temporelle inter-image est pleinement exploitée, cet aspect a un impact relativement faible sur les résultats de suivi.

Mesure de pertinence de la couleur. L'approche est inspirée des travaux de [23] concernant la détection de contours dans l'espace *HSV* en fusionnant les information de teinte, saturation et luminance à l'aide d'une fonction de pertinence dépendant de la saturation, typiquement une fonction sigmoïde. Dans notre cas, la fonction de coût utilisée dans l'approche de suivi par KLT est modifiée de manière à accorder une importance plus forte à L_1 lorsque la saturation est élevée et à la luminance sinon.

Résultats. Á partir d'expériences menées sur plusieurs séquences et en comparant les résultats de KLT appliqué dans différents espaces couleur, nous avons pu constater un très bon compromis entre le nombre de points correctement suivis et le temps d'exécution. Utilisé seul, l'invariant L_1 échoue lorsque la saturation est faible. La méthode proposant une association entre L_1 et la luminance I est capable de s'adapter à cette situation. Elle permet de maintenir les performances du suivi dans le cas de séquences saturées et améliore les résultats de manières significatives dans le cas de séquences peu saturées. D'autre part, utiliser les trois composantes (r, g, b) ne permet pas toujours de meilleurs résultats que L_1 mais augmente toujours, et de manière significative, les temps de calcul. Enfin, nous avons montré la pertinence du choix de L_1 comparé à la teinte pour l'ensemble des séquences de test. Ceci est probablement du au caractère bruité de cette composante.

0.4.2 Première coopération FoT + CT

Afin de répondre au problème du suivi d'objet sur de longues durées, nous proposons une approche fondée sur le suivi *FoT* évoqué dans la partie 0.3, qui analyse de manière robuste le flot optique associé aux points de l'objet. D'autre part nous utilisons le descripteur par covariance permettant de modéliser l'apparence globale (couleur et texture).

L'approche *FoT* estime de manière robuste le mouvement apparent de l'objet et permet théoriquement de résister aux occultations partielles jusqu'à 50% de masquage [146]. La restriction principale de cette méthode est que l'objet doit montrer un nombre suffisant de points saillants pour que le suivi par KLT soit fiable. Elle s'avère plus adaptée pour suivre des objets dont le mouvement est spatialement uniforme, autrement dit les objets rigides.

Le suivi par covariance *CT* apparaît comme une alternative qui de manière compacte modélise la cible par les corrélations au sein d'un ensemble d'attributs préalablement sélectionnés (communément il s'agit des coordonnées spatiales, de la couleur et de la couleur). Contrairement à *FoT*, de bonnes performances sont atteintes même pour des objets faiblement texturés ou pour des objets au mouvement non-rigide. Sa faiblesse principale est le temps d'exécution et ceci devient plus important lorsque la recherche de la nouvelle position de l'objet se fait de manière exhaustive dans l'image. Notons toutefois que *CT* peut être accélérée par un couplage avec d'autres méthodes de suivi telles que le filtrage particulaire [87] qui permet de limiter l'espace de recherche.

Le mécanisme proposé est décrit graphiquement par la figure 3-7. Tout d'abord, *FoT* est exécuté et l'analyse de la variance des vecteurs de mouvement permet d'identifier l'objet comme rigide ou non-rigide. S'il est rigide, *FoT* est exécuté tant que les résidus du suivi KLT (caractéristiques de l'erreur) restent inférieurs à une valeur seuil $\mathcal{R}_{max}$ ou que la dissimilarité

$\mathcal{D}_t$ entre deux matrices de covariance successives est inférieure à une valeur seuil $\mathcal{D}_{max}$. Si l'objet est non-rigide ou que *FoT* a échoué (à cause d'une occultation par exemple), *CT* est exécuté pour retrouver l'objet pendant une période de temps T. Si l'objet n'est pas retrouvé après cette période il est considéré comme définitivement perdu.

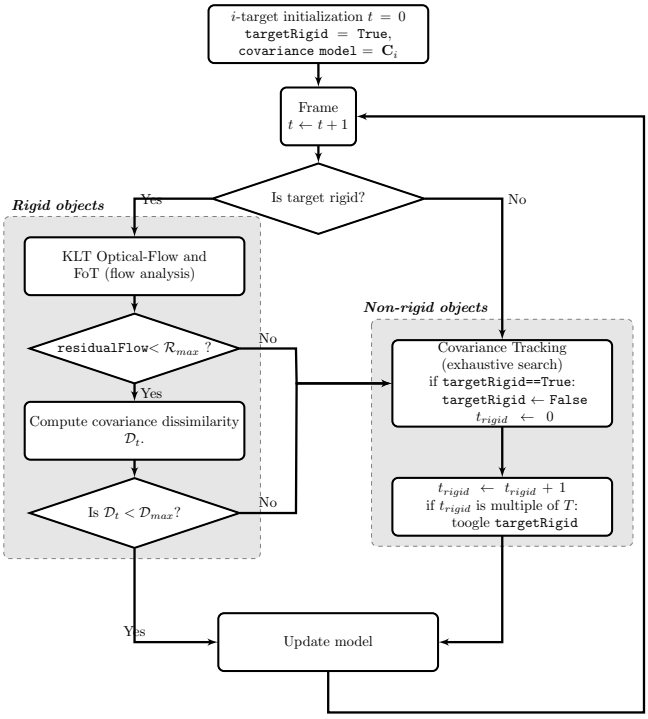


Figure 0-1: Mécanisme de coopération *FoT*+*CT*.

0.4.3 Deuxième coopération MS + CT

Les approces *MS* et *CT* possèdent également des avantages complémentaires. *MS* représente la cible par sa distribution couleur qui est quasiment insensible aux distorsions géométriques. *MS* est très rapide mais son pouvoir de séparation est relativement faible, ne permettant pas de faire face aux occultations et aux déplacements importants de l'objet dans l'image. Une association avec *CT* doit permettre de pallier cette faiblesse. Ici aussi la dissimilarité entre matrices de covariance est utilisée comme indicateur de précision de *MS* et l'algorithme *CT* est exécuté pour retrouver la cible après qu'elle soit perdue. La figure 3-12 illustre le mécanisme de commutation *MS*+*CT*.

Quatre séquences ont été utilisées lors des expérimentations. *Pedxing* et *Panda* sont des séquences où la caméra est quasiment fixe tandis que les séquences *Motocross* et *Carchase* correspondent à des vidéos prises d'une caméra mobile (soit fixée sur le cadre d'une moto, soit sur un hélicoptère). La tableau 3.3 analyse pour chacune d'entre elles le nombre de cycles par seconde (cpp) utilisé par chaque partie de l'algorithme de commutation *MS*+*CT* durant toute la durée de la séquence : *MS* et *CT* selon que *MS* ou *CT* est exécuté dans l'image, la mise à jour du modèle et le calcul du critère de similarité (qui est fait dans chaque image). Le pourcentage d'images pour lequel *MS* et *CT* sont sélectionnés par rapport à la durée totale de la séquence est également montré. Dans les séquences où la caméra est stable (*Pedxing* et *Panda*), *MS* est exécutée la plupart du temps étant donné que le mouvement est régulier. Lorsque l'objet n'est plus visible, *CT* est exécuté

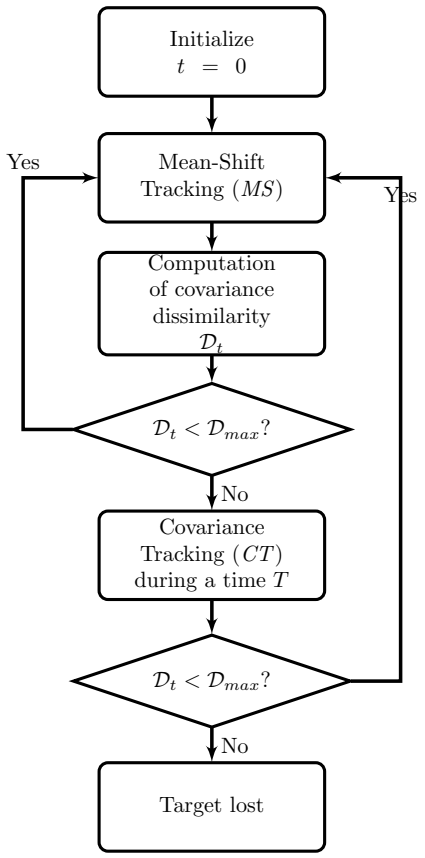


Figure 0-2: Mécanisme de commutation *MS*+*CT*.

pour retrouver la cible. Lorsque la caméra est placée sur un porteur mobile (*Motocross* et *Carchase*), le mouvement est complexe et imprédictible. Dans ce cas, *CT* est exécuté plus fréquemment.

0.4.4 Conclusion

Trois contributions ont été apportées en ce qui concerne le suivi adapté au contexte. Ces approches apportent l'un et/ou l'autre des avantages suivants :

1. *La robustesse*, soit par rapport aux occultations de l'objet, les mouvements abrupts, les changements de conditions d'acquisition.

2. *La rapidité*: dans la méthode de suivi adaptée aux changements de saturation, la couleur (qui requiert plus de temps de calcul) n'est utilisée que si nécessaire. Ensuite, les approches *MS* et *FoT* sont plus rapides mais moins robustes que *CT* seul. La combinaison de *CT* avec l'une des ces approches permet donc de réduire le temps d'exécution sans détériorer la qualité du suivi. Le suivi sera d'autant plus rapide que *MS* ou *FoT* seront appelées fréquemment. Cela correspond aux cas où la caméra est statique et que l'objet n'est pas occulté.

Table 1: Analyse de la procédure de coopération *MS+CT*. Ces résultats ont été obtenus sur un processeur *Nehalem*. Le tableau montre pour chaque séquence : la taille de l'objet suivi à l'initialisation ; le pourcentage d'images (%im) pour lequel les algorithmes *MS* et *CT* algorithmes sont exécutés ; le nombre total de cycles par pixel (*cpp*) passé par chaque partie de l'algorithme (*MS*, *CT*, mise à jour du modèle de covariance, calcul de la similarité) pendant toute la durée de la séquence.

Séquence	Taille	# im	*MS*		*CT*		M.à.J modèle	Similarité	Temps
	objet		*cpp* ($\times 10^6$)	%im	*cpp* ($\times 10^6$)	%im	*cpp* ($\times 10^6$)	(*cpp* $\times 10^6$)	[ms]/im
Pedxing	93×35	189	7.20	70	60.59	30	1.07	2.45	10.1
Panda	23×28	940	2.48	88	8.74	12	0.57	0.33	1.6
Motocross	64×47	2665	5.07	12	30.04	88	0.005	1.06	10.6
Carchase	45×97	2999	0.99	13	19.19	87	1.03	1.44	7.3

0.5 Suivi multiple temps-réel par covariance et ré-identification

L'un des objectifs de ce chapitre est de trouver la meilleure combinaison d'opérateurs de texture et de couleur qui améliorent à la fois la robustesse et le caractère distinctif du descripteur de covariance tout en gardant sa compacité et sa rapidité de calcul ainsi qu'une bonne robustesse vis-à vis des changements de couleur et d'illumination.

Ces matrices de covariance sont ensuite utilisées dans le cadre de la ré-identification d'objets et du suivi multiple. À cet effet, une méthode de suivi multi-objet originale est proposée, elle minimise une fonction d'énergie discrète combinant les probabilités d'association des trajectoires aux différentes cibles ainsi que leur concordance avec les modèles d'apparence décrits par des matrices de covariance.

0.5.1 Étude du choix des primitives texture

Plusieurs approches existent pour inclure des informations de texture au sein des matrices de covariance en vue d'améliorer leur pouvoir discriminant, citons *GRMC* (pour *Gabor Region Covariance Matrix* en anglais) utilisant les réponses issues des bancs de filtres de Gabor (voir partie 0.2.1) [104, 135], ce qui mène à une matrice de covariance de taille relativement importante et donc à des temps de mise en correspondance prohibitifs. Certaines publications remplacent les filtres de Gabor par des LBP (voir partie 0.2.1) comme dans la méthode *LBCM* (pour *Local Binary Covariance Matrix* en anglais) de [59], le *GLRCD* (pour *Gabor-LBP based Region Covariance Descriptor* en anglais) [154]). C'est également l'approche que nous avons privilégiée puisqu'elle répond à nos objectifs d'efficacité et de compacité.

Le choix de la *meilleure* façon de présenter l'information de texture sous la forme de motifs binaires locaux dans le descripteur de la matrice de covariance n'est pas évident à faire. *GLRCD* utilise une introduction brute des valeurs décimales des *LBP*s, valeurs très instables dans le cas des rotations. En outre, les opérations arithmétiques classiques ne sont plus applicables (ajouter ou sommer plusieurs valeurs décimales de *LBP* n'a pas de signification en termes de texture). *LBCM* utilise chaque bit dans le modèle de *LBP* comme un élément indépendant formant une chaîne de P bits. Ceci s'avère plus stable et il devient cohérent d'effectuer des opérations arithmétiques de façon indépendante pour chaque bit de la chaîne. Le problème est que le nombre de fonctions à l'intérieur de la matrice de covariance augmente avec le nombre P de bits considérés pour le codage des *LBP*s. Cela a un impact significatif sur la vitesse d'exécution .

Dans ces travaux de thèse, nous proposons une nouvelle façon d'introduire les LBPs dans les matrices de covariance

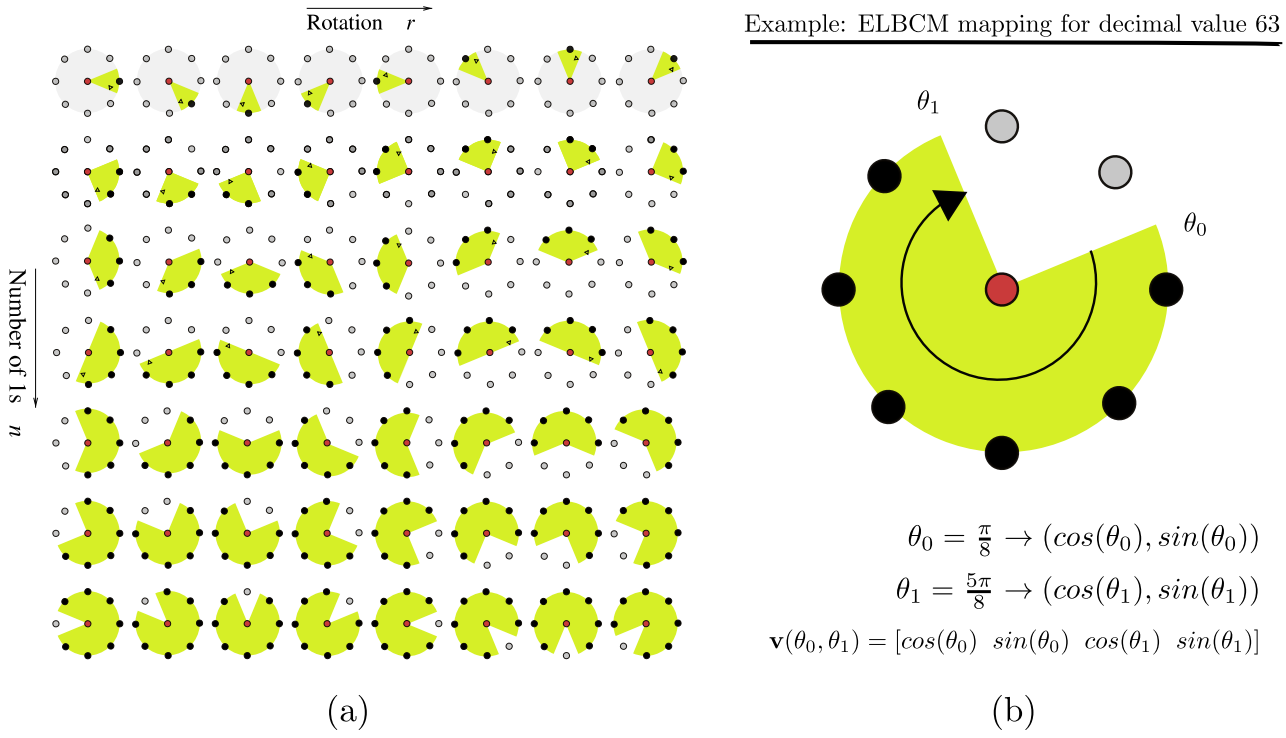


Figure 0-3: (a) Ensemble des 56 motifs *LBP* uniformes dans un voisinage $(8, R)$. Pour chaque motif, une flèche circulaire indique le début et la fin de l'angle (θ_0 and θ_1). (b) Dans la méthode *ELBCM*, un motif *LBP* défini par deux angles θ_0 et θ_1 est décrit par $\mathbf{v}(\theta_0, \theta_1) = [cos(\theta_0) \;\; sin(\theta_0) \;\; cos(\theta_1) \;\; sin(\theta_1)]$. Le motif *LBP* associé à la valeur décimale 63 prend la valeur $\mathbf{v}(\pi/8, 5\pi/8) = [cos(\pi/8) \;\; sin(\pi/8) \;\; cos(5\pi/8) \;\; sin(5\pi/8)]$.

par un nouveau descripteur de covariance appelé *ELBCM* (pour *Enhanced Local Binary Covariance Matrix*). Il utilise les angles définis par les motifs *LBP* uniformes présentés sur la figure 4-1. L'utilisation des formules trigonométriques est coûteuse en temps de calcul, mais ce problème est complètement résolu en utilisant une table de correspondance associant directement à chaque motif les valeurs de cosinus et sinus des angles θ_0 et θ_1.

ELBCM présente de multiples avantages par rapport aux précédents descripteurs de texture à base de covariance. Il est plus compact (7 composantes contre 11 pour *LBCM*) et plus stable car moins affecté par les petites rotations, qui impactent seulement les angles θ_0 et θ_1 alors que pour *LBCM* les mêmes rotations affectent irrégulièrement la valeur des bits dans le vecteur descripteur *LBP* en fonction de leur position. Un avantage supplémentaire du descripteur de la *ELBCM* est que sa taille est complètement indépendante du nombre de P voisins utilisés dans le modèle LBP.

L'apport de ELBCM a été montré sur trois applications : la classification de texture, la classification d'expressions faciales, le suivi d'objets. Les résultats obtenus démontrent la généricité de cette approche et les bonnes performances tant en termes de temps d'exécution qu'au niveau du pouvoir discriminant de la représentation.

0.5.2 Étude des primitives couleur

Après l'étude sur la texture, nous nous intéressons au choix des composantes couleur les plus pertinentes à introduire dans la matrice de covariance en vue d'une mise en correspondance efficace c'est-à-dire précise (qui produit peu d'ambiguïtés d'appariement) et rapide. Quatre représentations couleur sont utilisées : RGB, HSV, les modèles de couleur gaussiens

évoques dans la partie 0.2.1 et l'invariant L_1 introduit dans 0.4.1.

Deux applications sont visées : la classification de textures couleur et le suivi de visage dans le cas de conditions d'illumination variables. Les expérimentations réalisées dans le premier cas nous permettent de constater l'apport de l'approche $ELBCM$ par rapport aux approches comparables. Dans la seconde application, seules les combinaisons utilisant la composante invariant $L_1 = max(r, g, b)$ permet de résister aux forts changements d'illumination. Pour des conditions d'acquisition constantes, la méthode $ELBCM$ offre de bons résultats pour la plupart des espaces couleur utilisés.

0.5.3 Ré-identification d'objets

Principes. La mise en correspondance est rendue difficile lorsque les objets doivent être re-détectés et ré-identifiés à partir d'une seconde caméra sous des conditions d'acquisitions différentes en termes d'illumination et de perspective, à des instants très différents. Le descripteur de l'objet doit alors être suffisamment caractéristique de l'objet. À cet effet, Bak *et al*. ont proposé dans [6] un maillage dense de matrices de covariances [6] ainsi qu'une opération de moyenne appelé MRC (pour *Mean Riemannian Covariance* en anglais), qui permet de fusionner au cours du temps les informations d'apparences issues de ces multiples échantillons. En partant de cette approche, nous proposons de réduire le temps d'exécution par l'usage d'un nouvel arrangement de matrices *MRC*. Les régions d'intérêt associées aux objets sont redimensionnées (typiquement 96×128 pixels), puis des anneaux concentriques de rectangles sont positionnés autour du centre de l'objet, avec des aires qui augmentent de manière exponentielle pour permettre un recouvrement entre deux couches successives. Cela est inspiré des méthodes de FREAK et DAISY [1, 132] mais pour des régions rectangulaires, permettant des calculs rapides grâce à la méthode d'image intégrale.

Résultats. Quelques résultats sont visibles sur la figure 4-19. Afin de valider notre approche de ré-identification nous utilisons les courbes CMC (pour *Cumulative Matching Characteristic* en anglais) utilisées également dans [6] et [58]. Elles representent en ordonnée le pourcentage de fois où l'identité réelle apparaît parmi les n meilleurs appariements, avec n en abscisse. Ces tests sont réalisés sur les bases de données ETHZ [121] et PETS'09 L1-Walking-Sequence 1 [39]. Les taux de ré-identification obtenus sont comparables à ceux rapportés dans [6] mais en utilisant 75% de matrices de covariance en moins et des matrices plus compactes.

0.5.4 Suivi multiple

Le but de notre algorithme de suivi multi-objet est de détecter, d'identifier et de tracer les positions des objets. Idéalement le nombre de trajectoires par objet de un exactement, mais les inter-occlusions, les disparitions, l'apparition de faux positifs de la méthode de détection peut provoquer des dérives et des fragmentations des trajectoires. La possibilité d'inclure des informations relatives à l'apparence dans la fonction d'énergie a été laissée ouverte dans [3]. Cette idée est récupérée ici et enrichie en efficacité et en robustesse grâce à la représentation des cibles par les matrices de covariance.

Image par image l'algorithme évalue toutes les correspondances possibles entre les détections $\mathbf{d}_i$ et les trajectoires $\mathbf{p}_j$. De nouvelles pistes sont créées lorsque cela est nécessaire. Finalement, l'algorithme génère une cartographie dynamique contenant l'ensemble couples $M_T = \{(\mathbf{d}_i, \mathbf{p}_j)\}$ qui minimise une fonction d'énergie discrète. Cette fonction d'énergie $E(\mathbf{d}_i, textbfp_j)$ privilégie les configurations plausibles et pénalise celles qui sont incohérentes. Notre fonctionnelle d'énergie comprend la plupart des termes utilisés dans [3] : un terme de détection fondée sur la détection de données

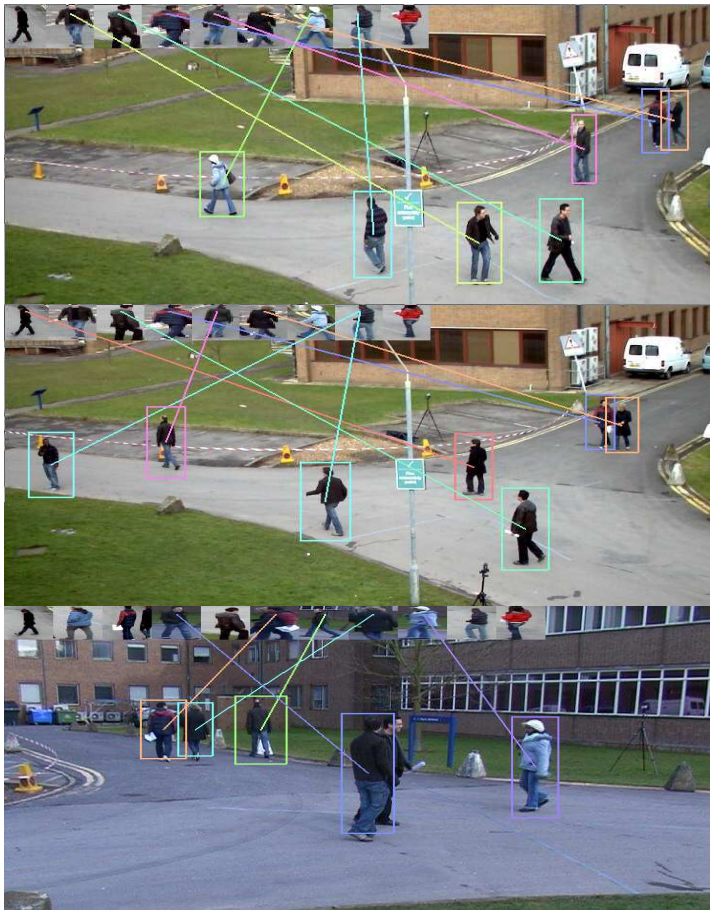


Figure 0-4: Exemples de résultats obtenus sur la base de données PETS'09, montrant une ré-identification à différents instants et à différents points de vue.

image E_{det}, un terme physique qui modélisant la dynamique des objets (vitesses linéaire et angulaire) E_{dyn} , un terme d' apparence E_{app} qui mesure la similarité entre la cible à l'emplacement actuel et le modèle de covariance associé aux trajectoires $\mathbf{p}_j$. Le dernier terme pénalise les créations (ou disparitions) de nouvelles pistes favorisant les trajectoires persistantes et continues E_{add}. Tous les termes mentionnés ci-dessus sont accumulés tel que l'exprime l'équation suivante

$$E(\mathbf{d}_i, \mathbf{p}_j) = \underbrace{E_{det}}_{\text{confiance de détection}} + \underbrace{E_{dyn}}_{\text{modèle dynamique}} + \underbrace{E_{app}}_{\text{similarité d'apparence}} + \underbrace{E_{add}}_{\text{modèle de persistance}} . \quad (1)$$

Une de nos contributions est de rajouter le terme d'énergie associé à l'apparence, celle-ci étant modélisée par les matrices de covariance. Ces matrices appartient à un espace Riemanien formé par l'ensemble des matrices définies positives (*SPD*). Une variété de métriques et de dissimilarités a été proposée dans la litérature afin de comparer deux matrices dans cet espace. Parmi elles, la mesure de dissimilarité *Jensen-Bregman LogDet Divergence* [29] s'avère particulièrement intéressante car elle est rapide à calculer et elle jouit de très bonnes propriétés de robustesse. Concernant la minimisation de (4.11), nous utilisons l'algorithm de Munkres [99] (la méthode Hongroise) utilisée par ailleurs dans [72].

Résultats. Nous avons évalué l'algorithme en utilisant les métriques proposées dans [127] et [83] et en partant de bases de données très largement utilisées dans la communauté ainsi que des séquences enregistrées dans notre laboratoire

en utilisant le premier point de vue : la base PETS 2009-S2L1-V1 [1], la base TUD [2] avec les séquences TUD-Campus, TUD-Crossings et TUD-Stadtmitte. La vérité-terrain est disponible pour toutes ces bases de données. Nos résultats sont visible en-ligne[3]. Sur la base de ces expériences, les résultats montrent que notre approche de suivi n'est pas aussi précise que celles proposées par [95] ou [12], mais elle a l'avantage important de travailler à la volée, en estimant les trajectoires image par image. Dans la méthode proposée dans [95], le suivi nécessite l'ensemble des images de la vidéo pour estimer le nombre de trajectoires optimal et leur tracé le plus probable.

0.5.5 Bilan

Dans cette partie, une étude a été menée pour proposer un nouveau descripteur par matrice de covariance en proposant une combinaison de composantes couleur et de texture, dans le but d'assurer un bon pouvoir discriminant tout en réduisant la taille de la représentation. Concernant la texture, le descripteur $ELBCM$ est proposé. Les résultats obtenus en classification de texture, suivi et ré-identification d'objets sont très prometteurs. Ce descripteur est également utilisé dans le cadre du suivi multi-cibles, où nous avons proposé une nouvelle fonction de coût intégrant un modèle d'apparence basé sur la divergence de Jensen-Bregman.

0.6 Implementation temps-réel: optimisations logicielles et transformations algorithmiques

Dans cette partie nous décrivons les transformations et accélérations opérées sur l'algorithme de suivi par covariance. Notons que ce travail a également été mené dans le cas du suivi *Mean-Shift* en vue de faire coopérer les deux méthodes [79].

Afin d'évaluer les performances des algorithmes et l'impact des optimisations, des comparaisons ont été faites sur trois générations de processeurs : Penryn/Yorfield, Nehalem/Bloomfield et Sandy-Bridge qui sont tous des processeurs multi-cœurs, excepté le Nehalem. Le processeur du Sandy-Bridge a la possibilité d'être *over-clocké*. La mémoire RAM peut être plus *over-clockée* que le processeur ce qui signifie qu'à une fréquence plus élevée (4.4 GHz) la RAM semble plus rapide ce qui résulte en un nombre réduits de nombre de cycles.

Deux stratégies peuvent être exploitées pour optimiser CT. La première consiste à réaliser du *multi-threading* en parallélisant la boucle principale de traitement. Cela est effectué avec OpenMP. La seconde nécessite une transformation de la mise en forme des données en mémoire SdM→MdS (*Structure de Matrices* vers *Matrices de structures*). Le but de cette manipulation est de transformer un ensemble de matrices indépendantes en une matrice unique, où chaque cellule combine les éléments de toutes les matrices dans une structure de données. La contribution d'une telle transformation est de tirer parti de la performance du cache en exploitant la localité spatiale et temporelle. L'algorithme de suivi par covariance se compose de trois parties :

1. le calcul des produits point à point pour toutes les primitives,

[1] http://www.cvg.rdg.ac.uk/PETS2009
[2] http://www.mis.tu-darmstadt.de/node/428
[3] http://andresromeromier.wikispaces.com/

2. le calcul de l'image intégrale des primitives,

3. le calcul de l'image intégrale des produits.

Le produit de caractéristiques et de sa transformation sont décrits dans les algorithmes 16 et 17. Les notations utilisées sont les suivantes :

- h et w : hauteur et largeur de l'image
- n_F : nombre de caractéristiques (primitives) image
- n_P : nombre de produits de caractéristiques, $n_P = n_F(n_F + 1)/2$,
- F : cube (*SdM*) ou matrice (*MdS*) de caractéristiques,
- P : cube (*SdM*) ou matrice (*MdS*) de produits de caractéristiques,
- I_F et I_P : deux cubes (ou matrices) d'images intégrales (à partir de F ou P).

Grâce à la commutativité de la multiplication, la moitié seulement des produits doivent être calculés (la boucle sur k_2 commence à k_1, ligne 3). Comme les deux dernières étapes sont similaires, nous ne présentons qu'une version générique du calcul de l'image intégrale (Algo. 18) et sa transformation (Algo. 19).

Algorithm 1: Optimisation ode CT - version *SdM* du produit de caractéristiques.

```
1 k ← 0
2 foreach k1 ∈ [0..nF − 1] do
3     foreach k2 ∈ [k1..nF − 1] do
4         [ point-to-point multiplication ]
5         foreach i ∈ [0..h − 1] do
6             foreach j ∈ [0..w − 1] do
7                 P[k][i][j] ← F[k1][i][j] × F[k2][i][j]
8                 k ← k + 1
```

Algorithm 2: Optimisation de CT - version *MdS* du produit de caractéristiques.

```
1 foreach i ∈ [0..h − 1] do
2     foreach j ∈ [0..w − 1] do
3         k ← 0
4         foreach k1 ∈ [0..nF − 1] do
5             foreach k2 ∈ [k1..nF − 1] do
6                 P[i][j × nP + k] ← F[i][j × nP + k] × F[i][j × nP + k]
7                 k ← k + 1
```

Une fois que cette transformation est faite, nous utilisons les instructions SIMD dans les différentes parties de l'algorithme. Pour ce qui concerne le produit, les deux boucles internes à k_1 et k_2 sont totalement déroulée de manière à montrer la liste de toutes les multiplications et la liste des vecteurs à construire par des instructions de permutations[4]. Par exemple, pour

[4] Cela est fait par `_mm_shuffle_ps` en SSE

Algorithm 3: Optimisation de CT - version *SdM* de l'image intégrale.

```
foreach k ∈ [0..n − 1] do
    [classical in place integral image]
    foreach i ∈ [0..h − 1] do
        foreach j ∈ [0..w − 1] do
            I[k][i][j] ← I[k][i][j] + I[k][i][j − 1] + I[k][i − 1][j] − I[k][i − 1][j − 1]
```

Algorithm 4: Optimisation de CT - version *MdS* de l'image intégrale.

```
foreach i ∈ [0..h − 1] do
    foreach j ∈ [0..w − 1] do
        foreach k ∈ [0..n − 1] do
            I[i][j×n+k] ← I[i][j×n+k] + I[i][(j−1)×n+k] + I[i−1][j×n+k] − I[k][i−1][(j−1)×n+k]
```

une valeur typique de $n_F = 7$, il y a $n_P = 28$ produits, elles sont effectuées dans l'ordre suivant (les nombres entre crochets sont les indices des caractéristiques) :

$$[0,0,0,0] \times [0,1,2,3]; \quad [0,0,0,1] \times [4,5,6,1];$$
$$[1,1,1,1] \times [2,3,4,5]; \quad [1,2,2,2] \times [6,2,3,4];$$
$$[2,2,3,3] \times [5,6,3,4]; \quad [3,3,4,4] \times [5,6,4,5];$$
$$[4,5,5,6] \times [6,5,6,6].$$

Dans ce cas, le septième vecteur est rempli à 100 %, mais il deviendra sous-optimal si n_P n'est pas divisible par le cardinal du vecteur (4 avec SSE, 8 avec AVX). Les permutations sont réalisées en une seule instruction pour certains d'entre eux, et en deux instructions dans le pire des cas. Parce que certaines permutations peuvent être réutilisées pour réaliser d'autres permutations, une factorisation est faite sur toutes les permutations nécessaires. Par exemple, avec $n_F = 7$, quinze remaniements sont nécessaires.

La quantité de mémoire requise par CT est égale à $(n_F + n_P) \times$ `sizeof(float))` $\times\, h \times w$ octets. En supposant $h = w = 1024$, $n_F = 7$ et $n_P = 28$, l'algorithme a besoin de 140 Mo, beaucoup plus que la taille de la plus grande mémoire cache disponible ! Afin d'évaluer l'impact des optimisations, les trois versions de l'algorithme (SdM, MdS, $MdS + SIMD$) ont été comparées pour des tailles variant de 128×12 à 1024×1024. Les temps d'exécution obtenus sont reportés dans la tableau 2.

Table 2: Optimisation de CT : temps d'exécution (ms) pour une image 1024×1024 avec 7 caractéristiques.

Processeur	*mono-threading*			*multi-threading*		
	SdM	MdS	MdS+SIMD	SdM	MdS	MdS+SIMD
Penryn	219.6	144.5	117.7	137.3	140.0	110.7
Bloomfield	219.3	118.2	65.5	91.0	116.5	69.9
SandyBridge	103.6	52.5	30.9	43.5	51.1	31.0
SandyBridge+OC	77.7	41.0	25.7	33.6	40.5	25.9
Nehalem	103.8	79.7	48.6	29.8	63.6	38.0

La transformation *SdM* $\rightarrow$ *MdS* est très efficace avec une accélération de près de $\times 2$. La version SIMD permet une accélération de $\times 2.5$ pour des images 128×128. Lorsque les données ne rentrent pas dans le cache (pour des images

1024×1024), cette accélération descend à une valeur moyenne de $\times 1,6$. Lorsqu'ils sont combinés ensemble, l'accélération atteint un maximum de $\times 5,11$ pour des tailles 128×128 et $\times 3,35$ pour des tailles 1024×1024. En ce qui concerne le Nehalem octo-cœurs, les accélérations sont plus faibles, mais le temps d'exécution est plus faible aussi, en raison de ses bus de cache rapides. Le gain apporté par OpenMP est moins important excepté pour Nehalem comparé au gain du codage SIMD. Concernant les temps d'exécution, la transformation *SdM*$\rightarrow$*MdS* + SIMD est nécessaire pour permettre une exécution temps-réel (40 ms par image) pour des images 1024×1024. Dans le cas de l'implémentation mono-thread, seul le SandyBridge (avec instructions SSE) est temps-réel et peut permettre un gain de $\times 4.0$ après augmentation de la fréquence d'horloge.

L'exécution de CT a été testée sur les séquences utilisées dans [79] à la fois en version monochrome et couleur. À cause des performances réduites des versions multi-threadées, l'algorithme est finalement implémenté et évalué sans multi-threading. Après les différentes transformations algorithmiques, les tests effectués montrent des temps d'exécution allant de 3 ms à 11ms, rendant donc l'algorithme rapide et compatible avec une exécution temps-réel sur un cœur.

Introduction

With the development of optics, electronics and computers, it became possible for science to capture, store and process great amounts of spatial information in the form of bi-dimensional or higher dimensional arrays. These technologies pushed scientists to think on the possibility of automatizing and emulating the visual perception abilities of animals and human beings. Most computer vision efforts are trying to automatize the ability to recognize objects, scenes and categories. An important variety of potential applications have been imagined *e.g.*, content-based image search, video data mining, object identification for mobile robots. Some of them already exist in a limited form *e.g.*, OCR characters, car plates and finger print recognition.

Visual recognition is the problem of learning visual categories from a limited set of samples and identifying new instances of those categories, the problem is often separated into two types: the specific case and the generic category case. In the specific case the objective is to identify instances of a particular object, place or person *e.g.*, the portrait of a magazine, people registered in a database and popular monuments such as the Eiffel tower. Whereas in the generic category case we seek to recognize different instances that belong to the same conceptual class *e.g.*, cars, pedestrians, road signs and mugs. Specific object recognition works by matching and geometric verification. In contrast, generic object categorization often includes a statistical model of their appearance and/or shape. Learning visual objects requires to gather a set of training images and to extract a model able to identify the presence and/or localization of instances in novel images. The type of training data that is required depends on the desired detail-level of recognition i.e., *categorize* objects present in the image, *detect* them with a coarse spatial localization, or to *segment* the pixels belonging each of the objects and the clutter from background.

Visual object matching and modeling is challenging because the instances of the same object can appear very different due to many often uncontrollable variables such as illumination conditions, camera viewpoint, object pose, partial occlusions and clutter from the background. Generic category recognition can be even more challenging as different object instances from the same category often exhibit significant variations in appearance.

This thesis proposes a computer vision system for detecting and tracking multiple targets in videos.

The covariance matching method, as defined by Porikli *et al*. [109], is the guiding thread of our work because it offers a compact representation of the target by embedding heterogeneous features in a elegant way. Therefore, it is efficient both for tracking and recognition.

Four categories of contributions are proposed.

The first one deals with the adaptation to a changing context, following two aspects. A preliminary work consists in the adaptation of color according to lighting variations and relevance of the color. Then, literature shows a wide variety of tracking methods, which have both advantages and limitations, depending on the object to track and the context. Here, a deterministic method is developed to automatically adapt the tracking method to the context through the cooperation of two complementary techniques. A first proposition combines covariance matching for modeling characteristics texture-color information with optical flow (KLT) of a set of points uniformly distributed on the object [113]. A second technique associates covariance and Mean-Shift. In both cases, the cooperation allows a good robustness of the tracking whatever

the nature of the target, while reducing the global execution times [79].

The second contribution is the definition of descriptors both discriminative and compact to be included in the target representation. To improve the ability of visual recognition of descriptors two approaches are proposed. The first is an adaptation operators (LBP to Local Binary Patterns) for inclusion in the covariance matrices . This method is called ELBCM for Enhanced Local Binary Covariance Matrices [115]. The second approach is based on the analysis of different spaces and color invariants to obtain a descriptor which is discriminating and robust to illumination changes. The various experiments implemented in tracking and recognition (texture , faces , pedestrians) show very promising results.

The third contribution addresses the problem of multi-target tracking, the difficulties of which are the matching ambiguities, the occlusions, the merging and division of trajectories. We also propose the re-identification of targets using a set of spatially adapted covariance descriptors and minimizing a function of discrete energy that takes into account the kinematic behavior of the whole objects and model their appearance [116].

Finally to speed algorithms and provide a usable quick solution in embedded applications this thesis proposes a series of optimizations to accelerate the matching using covariance matrices. Data layout transformations, vectorizing the calculations (using SIMD instructions) and some loop transformations had made possible the real-time execution of the algorithm not only on Intel classic but also on embedded platforms (ARM Cortex A9 and Intel U9300) [117].

The first two chapters of this thesis provide an overall panorama of the state of the art techniques, the following two chapters describe all the contributions to the fields of image processing and computer vision. The last chapter is consecrated to the optimization and acceleration of the covariance tracking algorithm.

Chapter 1 covers some of the most common local pixel information operators used to detect and analyze abrupt and transitional changes (*e.g.*, color, gradients, edges, corners, contours) for a variety of image pattern recognition applications such as image segmentation, key-point detection/matching and optical flow. All these techniques are the base of higher-level computer vision algorithms that contemplate the target shape, color and texture patterns to perform object detection,recognition and tracking. This thesis is primarily focused on the improvement of the covariance region descriptor.

Some of the state-of-the-art techniques for the detection and matching of objects in video sequences are introduced in Chapter 2. The detection part of the chapter is composed by very brief introductions to some of the most important classifiers: Support Vector Machines (SVM's), AdaBoost and classification with Riemannian manifolds (the type of topological spaces where covariance matrices reside). The second part of the chapter is devoted to matching, the difference here is that the objective is not to find a generic category of objects but to match particular instances based on their appearance. Chapter 3 is dedicated to the long-term tracking problem. It explores some of the situations where a particular tracking method fails while others succeed. A robust algorithm could result from the combination of an ensemble of methods if a mechanism is employed to find an agreement between the algorithms so that the successful ones compensate the failures of the others. Different approaches can be used to combine a set of tracking algorithms: a competition scheme selects the best output from the set of tracking method responses; a cooperation scheme structured in the form of a cascade applies first the least expensive and less accurate one which finds a cheap but rough estimate of the target location, the intermediate results are then refined by more precise (but consuming) methods. A different alternative is to analyze the context and select the algorithm that best adapts to the situation. This contextual switching approach is thoroughly explored in this chapter. To finalize the contributions to the field of computer vision Chapter 4 presents a series of modifications to increase the discriminant power of the covariance descriptor by integrating some textural and color operators. Textural information

is incorporated by means of local binary patterns (LBP). Color is processed using invariants for brightness and saturation changes. Evaluations of these improvements for object tracking and other related problems such as texture analysis and facial gesture recognition are provided. Other improvements presented in this chapter are dedicated to the multiple object tracking and description problem. Two methods are proposed to face this problem: a pedestrian re-identification method based on an array (or grid) of covariance matrices, and a heuristic for the minimization of a discrete energy function that measures the complexity in the set of possible trajectories measuring its curvature, speed and appearance. The appearance similarity measure used to compare a pair of covariance matrices is the total Bregman divergence which is very efficient and fast to compute. This method implies a temporal coherency within the frames allowing the evaluation and fitting of curves that trace the target trajectories. The grid of covariance matrices is suitable for recognizing targets captured in a multiple-camera configuration, while the trajectory and appearance modeling of the second approach is best adapted for identifying the target trajectories on a single camera (preferably a fixed one) using a tracking by detection strategy. As mentioned above, Chapter 5 discusses the accelerations techniques proposed to implement the algorithms of this thesis in real-time on Intel desktop processors and other architectures such as the ARM Cortex 9. These methods are discussed in depth, proposing dedicated methods to increment the degree of parallelism of the algorithm mostly on the data level using data layout transformations together with SIMD instructions. At the end some multi-threading techniques are evaluated. The discussion about the global results and perspectives evoked by this work are discussed in final conclusions chapter. The first two chapters of this thesis provide an overall panorama of the state of the art techniques, the following two chapters describe all the contributions to the fields of image processing and computer vision. The last chapter is consecrated to the optimization and acceleration of the covariance tracking algorithm.

Chapter 1 covers some of the most common local pixel information operators used to detect and analyze abrupt and transitional changes (*e.g.*, color, gradients, edges, corners, contours) for a variety of image pattern recognition applications such as image segmentation, key-point detection/matching and optical flow. All these techniques are the base of higher-level computer vision algorithms that contemplate the target shape, color and texture patterns to perform object detection, recognition and tracking.

Some of the state-of-the-art techniques for the detection and matching of objects in video sequences are introduced in **Chapter 2**. The *detection* part of the chapter is composed by very brief introductions to some of the most important classifiers: Support Vector Machines (*SVM*'s), AdaBoost and classification with Riemannian manifolds (the type of topological spaces where covariance matrices reside). The second part of the chapter is devoted to matching, the difference here is that the objective is not to find a generic category of objects but to match particular instances based on their appearance.

Chapter 3 is dedicated to the long-term tracking problem. It explores some of the situations where a particular tracking method fails while others succeed. A robust algorithm could result from the combination of an ensemble of methods if a mechanism is employed to find an agreement between the algorithms so that the successful ones compensate the failures of the others. Different approaches can be used to combine a set of tracking algorithms: a competition scheme selects the best output from the set of tracking method responses; a cooperation scheme structured in the form of a cascade applies first the least expensive and less accurate one which finds a cheap but rough estimate of the target location, the intermediate results are then refined by more precise (but consuming) methods. A different alternative is to analyze the context and select the algorithm that best adapts to the situation. This contextual switching approach is thoroughly explored in this chapter.

To finalize the contributions to the field of computer vision **Chapter 4** presents a series of modifications to increase

the discriminant power of the covariance descriptor by integrating some textural and color operators. Textural information is incorporated by means of local binary patterns (LBP). Color is processed using invariants for brightness and saturation changes. Evaluations of these improvements for object tracking and other related problems such as texture analysis and facial gesture recognition are provided. Other improvements presented in this chapter are dedicated to the multiple object tracking and description problem. Two methods are proposed to face this problem: a pedestrian re-identification method based on an array (or grid) of covariance matrices, and a heuristic for the minimization of a discrete energy function that measures the complexity in the set of possible trajectories measuring its curvature, speed and appearance. The appearance similarity measure used to compare a pair of covariance matrices is the total Bregman divergence which is very efficient and fast to compute. This method implies a temporal coherency within the frames allowing the evaluation and fitting of curves that trace the target trajectories. The grid of covariance matrices is suitable for recognizing targets captured in a multiple-camera configuration, while the trajectory and appearance modeling of the second approach is best adapted for identifying the target trajectories on a single camera (preferably a fixed one) using a tracking by detection strategy

As mentioned above, **Chapter 5** discusses the accelerations techniques proposed to implement the algorithms of this thesis in real-time on Intel desktop processors and other architectures such as the ARM Cortex 9. These methods are discussed in depth, proposing dedicated methods to increment the degree of parallelism of the algorithm mostly on the data level using data layout transformations together with SIMD instructions. At the end some multi-threading techniques are evaluated. The discussion about the global results and perspectives evoked by this work are discussed in final conclusions chapter.

CHAPTER 1

Local information and object representations

Contents

Introduction

Computer vision is the field of signal processing that studies the information contained by bi-dimensional (or higher-dimensional) arrays of visual information looking for an interpretation expressed by decisions. In its origins in the 1960's, the objective of computer vision was to imitate the human vision and replicate it on the machines so that they could observe the scenes in the same way that humans do. But other domains have emerged since then using sensors of different nature than humans eyes, this is the case of microscopic images, medical imaging such as x-rays and magnetic resonance (MRI) scans, so, it is worth noting that by images we refer not only to the traditional photography formed by light in the human visible spectra (colors from red to violet), but to a broader category of images that includes any type of spatial evolving physical measurement can be treated: electromagnetic waves (light), acoustic waves (ultrasound imaging), magnetic resonance, etc.

At the beginning of the processing chain what computers only know is the image dimensions (width and height) and the type of spatial data stored in the form of pixels (*e.g.*, surface brightness, color, magnetic resonance intensity). Computer vision operates at different levels of abstraction discovering meaning inside these large arrays of numbers by comparing pixels with their neighbors with simple arithmetic operations like numerical differentiation, measuring interesting properties such as the direction of steepest change (*i.e.*, gradient vector) and its strength (contrast). Gradients are helpful to explore interesting local properties such as saliency which defines when some points are unique and deserve more interest. If these points or regions are stable enough their location on novel images taken at different instants or perspectives should be easy to establish. Following this concept, objects are described as clouds of connected interest points. Color provides a complementary source of information to gradients and texture, it allows us to recognize non-rigid (animated) objects such as animals and people by their characteristic colors appearances (given by their skin, hair, plumages or clothes). Furthermore, a variety of color invariants and constancy methods exist that allow us to handle illumination changes better for color than for brightness images. Because both sources of information are complementary, it is possible to use them together in a variety of applications such as image segmentation where the objective is to subdivide an image into multiple regions of homogeneous properties.

The evolution of images over time is also an essential source of information, pixels (or regions of them) moving in similar directions can be clustered together and separated from the rest. To detect changes over time, some methods do simple image subtraction (supposing a static camera configuration), others construct a statistical distribution explaining each individual pixel (or regions of them) by their distribution of brightness or color (using mixtures of Gaussian models or histograms). In any case, when pixel values are far from the modeled ones this can be an indicator of the presence of movement. This principle is widely used to detect moving objects in tracking and activity recognition applications. Motion images is also used for optical flow analysis which allows to describe patterns of visual motion related to objects, surfaces and edges. Higher-level applications such as shape from motion or camera stabilization.

Pattern analysis is often exploited to identify the presence of an object or an interesting property. Patterns are always associated to a descriptor used for comparing and matching. The quality of a pattern descriptor is measured in terms of three properties: compactness, distinctiveness and repeatability.

This chapter explores some of the very basic image processing techniques frequently used in computer vision to identify interesting points/features on images, how to build descriptors of them and the diversity of object representations currently used in the state of the art. A variety of topics related to the nature of the information contained in pixels (*e.g.*, brightness, color, motion and depth) and their neighbors is explored in the first section:

- Filter operators measuring inter-pixel relations.
- Histograms for representing distributions of pixel values: brightness, colors, gradients, etc.
- Textures, filter-banks, wavelets, local binary patterns and their statistical methods.
- Local features detectors (*e.g.*, Harris corners, *SIFT* and *SURF* points) and their descriptors.

1.1 Pixel information

Individual pixels store very few information. It is unconceivable to assign an interpretation to large arrays of data (containing perhaps millions of values) without considering their spatial and time relations and constraints. In addition, individual pixel values are unstable and vulnerable to noise. Image processing and computer vision methods analyze pixel neighborhoods to detect changes and their distributions in space and time to study their statistical properties and identify how local patterns are composed. Furthermore, a group of pixels related in time and/or space deserves more confidence than isolated pixels because in most of the cases noise doesn't conspire and can be easily filtered out by local linear and non-linear operators. Depending on the type of sensor used to capture the image pixel values can represent a) illumination intensity (brightness), b) a wavelength in the electromagnetic spectra and c) the distance from the focal point to the object object in the 3D space popularly known as *range information* or *depth*. Local arithmetic operations between pixel neighbors (in time and space) generate other types of images such as gradient, edges and motion images. Examples from all these kinds of images are shown in **Figure 1-1**. They are classified here into five distinct categories: lightness and color images, gradient and textural information, feature points and motion.

1.1.1 Brightness and color images

Natural images are formed by the physical interaction of three fundamental processes: illumination, material reflection and detection/observation. Depending on theses three elements a final gray-scale or color image is formed. Color and luminance images both are the result of the interaction of light, materials and the observers sensitivity.

The whole process starts with light illuminating the visual scene[1]. Many different attributes characterize a light source the radiation pattern and the spectra of photons over the wavelengths. When the proportion of short-wavelength photons in the emitted light is bigger than long-wavelength ones light looks bluish. In contrast, when more photons of long wavelengths are emitted, the color is reddish. Candle lights and halogen lamps emit smooth spectra that can be easily and uniquely characterized by a single number referred to as *temperature of radiation*.

Materials are the second aspect to think of in the process of image formation. Depending on their spectral reflectance, their geometry and their rugosity, materials modify the light beam spectrally (by absorbing some wavelengths) and geometrically by diffusion, specular reflection or refraction for instance.

The third process in image formation is how light is observed or recorded by a camera or the eye integration the photons energy over a certain bandwidth, spatial area and for a period of time. In the eye, the spectral integration is performed grouping the visual spectrum into three spectral broadbands (*i.e.*, short, middle and long-range wavelengths). The integration time takes around 50 ms and it is more sensible (acute) at the central area of the retina (the *fovea*) and less sensible in the periphery. Color cameras mimic this temporal and spectral characteristics of the eye, recording three color bands in about the same period of time. In most of the cases spatial resolution is uniform over the whole image, with a resolution in the range of mega-pixels.

[1] An electromagnetic radiation of a certain intensity composed by photons containing energy of certain wavelengths and traveling in a certain direction.

(a) (b) (c) (d) (e)

Figure 1-1: Pixel interpretation is context-dependent: (a) illumination intensity, (b) color components (*RGB*), (c) spatial gradient (Sobel), (d) motion: color denotes angle while represents speed and (e) the distance from the camera focal point (depth).

Brightness images

Surface brightness depends on how much incident light arrives at that patch and on the fraction of the incident light that gets reflected. A brightness pixel is influenced by three major phenomena: the camera response to light, the fraction of light reflected from the surface to the camera, and the amount of light coming to the surface. If $\mathbf{X}$ is a point in the space that projects to the point $\mathbf{x}$ in the image, and $I_{patch}(\mathbf{X})$ is the intensity emmitted by the surface at $\mathbf{X}$ and $I_{camera}(\mathbf{x})$ is the camera response at $\mathbf{x}$, our model is

$$I_{camera}(\mathbf{x}) = kI_{patch}(\mathbf{X}) \tag{1.1}$$

where k is a constant which in practice can be determined by calibration. Modern cameras have linear responses within a reasonable intensity range but pronounced non-linearities are expected for darker and brighter illumination conditions.

The amount of light received by a patch depends on the intensity of the light source (*luminaire*) and on the geometry of the scene. The amount of light depends on the angle formed by the surface normal and the source light as those patches facing the light collect more radiation than tilted away surfaces. Surfaces reflect light by a process of diffuse reflection which scatters light evenly across all the directions leaving the surface, as such, the perceived brightness of a surface should not depend on the viewing direction. Some fraction of light is usually absorbed by the surface, this absorption is independent from the incident light direction, a parameter which determines how much incoming light is reflected is the *albedo* and it represents the fraction of light which is reflected, diffuse surfaces are also known as *Lambertian*.

As mirrors are not diffuse, what an observer sees depends on the direction at which it looks to the mirror, this is phenomenon is referred as *specular reflections*.

More formally, the brightness of a pixel in a Lambertian surface (where light is equally reflected in all directions) is modeled by the amount of light a surface collects, this quantity is higher for surfaces facing the source light than for surfaces oriented along the direction in which rays travel. Mathematically, this is expressed by the cosine of the angle θ formed by the illumination ray and the surface normal,

$$I = \rho I_0 cos\theta, \tag{1.2}$$

where I_0 is the intensity of the light source, θ is the angle between the source direction and the surface normal, and ρ is the diffuse *albedo*. More complete models consider surfaces having diffuse and specular reflections, this is the *lambertian+specular model*, assuming an infinitely distant light source, we write $\mathbf{N}(\mathbf{x})$ for the surface normal at $\mathbf{x}$, $\mathbf{S}$ is a vector pointing from $\mathbf{x}$ towards the source with length I_0 (source intensity), $\rho(\mathbf{x})$ is the *albedo* at $\mathbf{x}$, and $Vis(\mathbf{S}, \mathbf{x})$ is a function

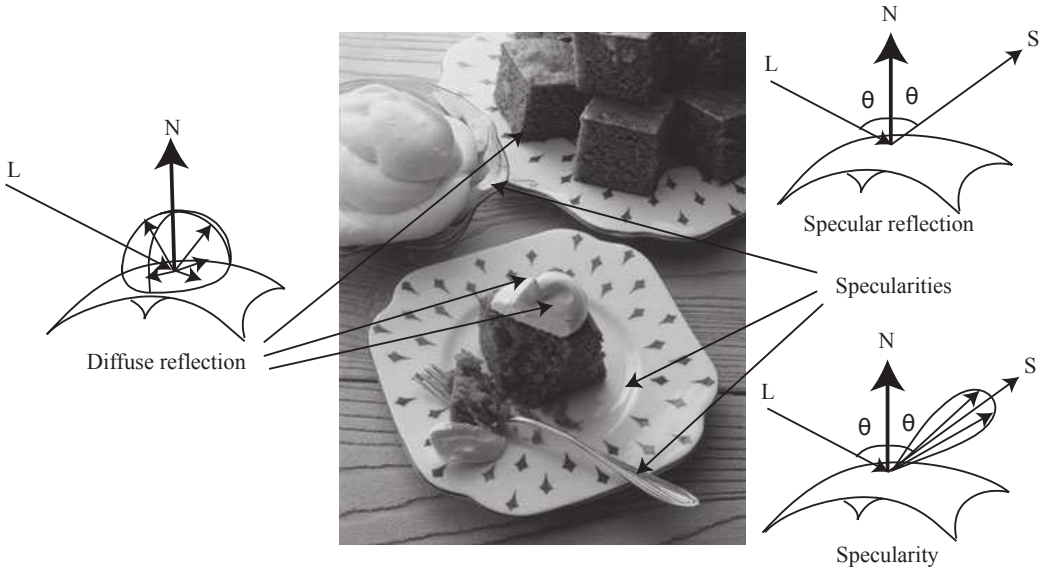


Figure 1-2: Diffuse (or Lambertian) and specular models are exemplified here. In the Lambertian model, the incoming light is diffused equally in all the observing directions, while in the specular reflection model light is reflected with the same angle θ formed by the direction of the incident ray of light and the normal to the surface at the point of incidence. In reality reflections are not purely Lambertian neither specular, and light is concentrated and diffused in some directions around the specular reflection direction. Image taken from [45].

that is 1 when **x** can see the source (when there is line of view) and zero otherwise. The intensity at **x** is

$$\underbrace{I(\mathbf{x})}_{\text{Image intensity}} = \underbrace{\rho(\mathbf{x})(\mathbf{N}\cdot\mathbf{S})Vis(\mathbf{S},\mathbf{x})}_{\text{Diffuse term}} + \underbrace{\rho(\mathbf{x})A}_{\text{Ambient term}} + \underbrace{M}_{\text{Specular term}} .$$

Interesting properties about the scene can be observed directly from brightness images *e.g.*, specularities are a source of information about the surface shape, the *albedo* is a inherent property of a surface rather than a property of the picture of it. Surface properties that do not change when image circumstances do are called intrinsic properties. The problem is that different *albedo* surfaces under different illuminations may report the same pixel intensity. *Lightness constancy* is a desirable property of a vision system, because it allows it to compensate changes in the intensity of illumination and accurately report the lightness of a surface (whether it is white, gray or dark).

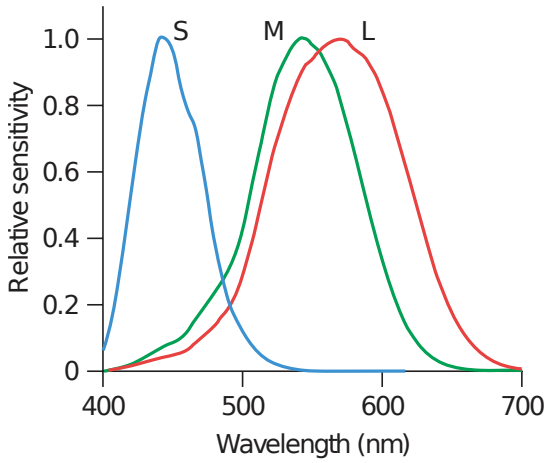


Figure 1-3: In the first row mosaics showing the cone mosaic at the central fovea of L-cones, M-cones and S-cones respectively. In the second row a graph displays the relative spectral sensitivity of the three cone types (figure taken from [21]).

Color images

Human vision is basically trichromatic: it is based on the responses of three different cone photo-receptors types located in our eyes retina. Ganglion cells in our retina combine the responses arising from these cones to form three opponent channels: an achromatic one (black and white) and two chromatic ones (red-green and yellow-blue). These information is sent in the form of pulse-like signals to the visual cortex, where perception of color takes place. However, the most abundant light-sensitive cells are the rods which are not sensitive to color and generate luminance intensity signals only. Each retina holds about one hundred million of photo-receptors and about 95% of them are rods. Depending on the light intensity level our vision adapts and works in three different states: *scotopic*, *mesopic* and *photopic*. Pure scotopic vision works in the light-dark dimension only, it is activated at very low light levels ($< 0.01 cd/m^2$). Mesopic vision corresponds to intermediate light levels ($0.01 - 1 cd/m^2$) in this state both rods and cones are active but color discrimination is very poor. Truly color discrimination is possible at light levels above ($1 cd/m^2$) when our vision becomes photopic. In the retina there are three different photo-receptors: L-cones, M-cones and S-cones. **Figure 1-3** represents the spectral sensitivities of each type of cone. It is worth noticing that the L and M-cones wavelength responses are largely overlapping while S-cones wavelength response is somehow isolated. At each wavelength there exists a unique combination of L, M and S sensitivities. The resulting response corresponds to an integral of the wavelength-by-wavelength product of the light spectrum arriving to the eye and the spectral sensitivity around the spectral window, resulting in a tuple formed by the three cone-type integrals. The perceived color is determined relatively to this 3-valued tuple, but not exclusively, as the visual systems also makes spatial comparisons, making the final perceived color dependent on the neighbor colors as well.

Besides modeling the reflections, computer methods need also to quantize the color information and represent it numerically. Any arbitrary numbering scheme could do the job, as long as each number uniquely defines a color. Colors could be represented by ordering sequentially all the colors as they appear in the rainbow, going from the deep red till the deep violet. For historical reasons, most of the cameras yield color information in an *RGB* scheme. But color information can be mapped to alternative schemes or color-spaces (*e.g.* *XYZ*, *RGB*, oponent color systems, L*u*v*, L*a*b*, *HSV*,

HSI, and *HSL*) which might appreciate better certain properties of color information than other schemes. For example, the hue-saturation-value model (*HSV*) decomposes the *RGB* values into an orthogonal coloring scheme, decoupling the intensity information from the chromatic information. Many computer vision tasks such as image segmentation and object recognition require stable and repeatable color measurements. To achieve this purpose, color invariant is a basic requirement. The desired color invariance is obtained by color transformations at a pixel level. Many of these transformations are non-linear and tend to intensify the amount of noise *i.e.* a small perturbation of the *RGB* values causes a large jump in the transformed values. Thus, it is necessary to study how noise amplification propagates in *RGB* values to propose stable algorithms adapted to the color information at hand.

Many luminance-based algorithms can be ported to the color domain, but color image processing methods should avoid the introductions of false chromaticities, in differential-based algorithms, the derivatives of the separate channels must be combined without loss of derivative information. The simple application of existing luminance-based operators on the separate color channels, and the subsequent combination fail because the influence of undesired artifacts. Computation of color gradients is performed better by color-tensor-based techniques [142] for example. The influence of color in perception and computer vision is huge, but this is still a developing field, and many questions and problems are still waiting a solution.

There are many color representations and models in literature, as in [136] and [21]. A different theory of color measurement called Gaussian color models is introduced here.

Gaussian color models

Image formation implies physical measurements over the spectral, spatial and time dimensions. Gaussian color models are a new theory of color measurement, it is an extension of the Gaussian derivative framework (scale-space theory) to the spatio-spectral domain [51]. One of the most important achievements in scale-space theory is the remark that Gaussian shapes prevent the creation of extra details on higher scale images (*i.e.* lower resolution). Gaussians offer a general probe for spatio-spectral differential quotients. Let $E(\lambda)$ be the spectral energy distribution of light, it is a function of λ, which denotes the wavelength. Let now $G(\lambda_0, \sigma_\lambda)$ be a Gaussian kernel of spectral scale σ_λ and positioned at λ_0. The spectral energy distribution $E(\lambda)$ may be approximated by the Taylor expansion at λ_0

$$E(\lambda) = E^{\lambda_0} + \lambda E^{\lambda_0}_{\lambda} + \frac{1}{2}\lambda^2 E^{\lambda_0}_{\lambda\lambda} + \cdots \tag{1.3}$$

Assuming measurements of the spectral energy distribution are made with a weighted integration over the spectrum we have

$$E^{\sigma_\lambda} = E^{\lambda_0,\sigma_\lambda} + \lambda E^{\lambda_0,\sigma_\lambda}_{\lambda} + \frac{1}{2}\lambda^2 E^{\lambda_0,\sigma_\lambda}_{\lambda\lambda} + \cdots \tag{1.4}$$

where $E^{\lambda_0,\sigma_\lambda} = \int E(\lambda)G(\lambda;\lambda_0,\sigma_\lambda)d\lambda$ measures the spectral intensity. Gaussian color models measure the the coefficients $E^{\lambda_0,\sigma_\lambda}$,$E^{\lambda_0,\sigma_\lambda}_{\lambda}$ and $E^{\lambda_0,\sigma_\lambda}_{\lambda\lambda}$ in the Taylor expansion of the Gaussian weighted spectral energy distribution at λ_0 with the spectral aperture (scale) σ_λ.

Spatial information can be introduced to the model yielding a Taylor expansion at wavelength λ_0 and coordinates

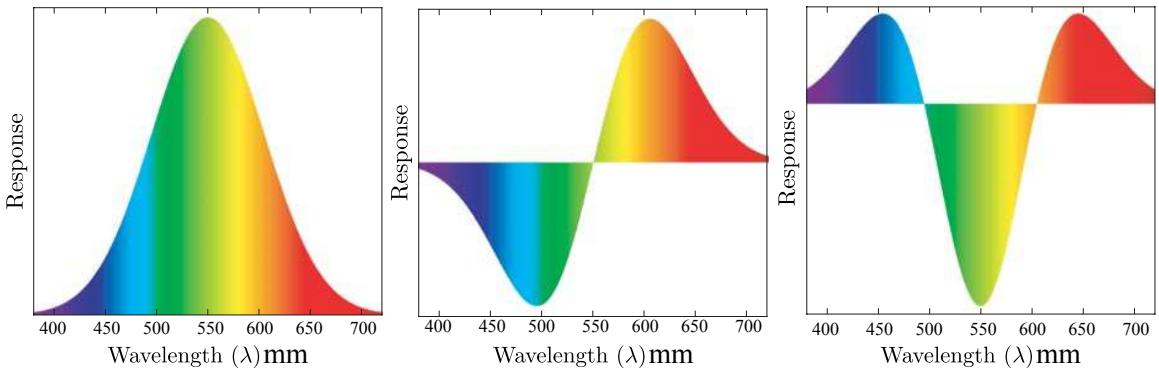


Figure 1-4: Gaussian color model spectral derivative responses E, E_λ and $E_{\lambda\lambda}$. Taken from [21].

$\mathbf{x} = (x, y)$. The result of this is the exploration of an energy density volume in a spatio-spectral space

$$E(\lambda, \mathbf{x}) = E + \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix}^T \begin{bmatrix} E_{\mathbf{x}} \\ E_\lambda \end{bmatrix} + \frac{1}{2} \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix}^T \begin{bmatrix} E_{\mathbf{xx}} & E_{\mathbf{x}\lambda} \\ E_{\lambda\mathbf{x}} & E_{\lambda\lambda} \end{bmatrix} \begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} + \cdots, \tag{1.5}$$

where $E_{\lambda^m \mathbf{x}^n}(\lambda, \mathbf{x}) = E(\lambda, \mathbf{x}) * G_{\lambda^m, \mathbf{x}^n}(\lambda, \mathbf{x}; \sigma_\lambda, \sigma_{\mathbf{x}})$ is the m-th differentiation with respect to λ (spectral differentiation) and the n-th with respect to $\mathbf{x}$ (spatial differentiation).

It is safe to assume that camera sensitivities approximate Gaussian functions around the red, green, and blue areas of the visible spectrum. An approximation of the Gaussian color model is given by the opponent color spaces. Here, the brightness or intensity channel $I = R+G+B$ can be regarded as a Gaussian-weighted spectral response, the yellow-blue channel $YB = R + G - 2B$ approximates the first-order spectral derivative because it results from comparing one half of the spectrum (around the blue color) with the other half (around yellow), the red-green channel $RG = R - 2G + B$ can be interpreted as a second-order derivative which compares the center of the spectrum with the extremes (in the visual bandwidth). The responses of the E, E_λ and $E_{\lambda\lambda}$ responses are shown in **Figure 1-4**. The opponent color interpretation of the Gaussian color model is then expressed as

$$\begin{bmatrix} \hat{E} \\ \hat{E}_\lambda \\ \hat{E}_{\lambda\lambda} \end{bmatrix} = \frac{1}{3} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -2 \\ 1 & -2 & 1 \end{pmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{1.6}$$

This color space is used further for color gradient computation and other matching applications in **Section 4.2**. More details about the Gaussian color models can be found in [21].

1.1.2 Gradients and edges

Image derivatives are essential to describe images local structure and their applications in computer vision are vast: edge and contour detection, feature extraction, optical flow, shape from shading, image segmentation and object detection. First order derivatives, gradients and edge images result from sharpness or contrast operators applied to gray-scale or color images. Sharp brightness changes are important indicators of information and a multiplicity of phenomena can cause those changes (*e.g.*, changes in surface normal direction and *albedo*, object and background occluding contours). Identifying the nature of this changes is valuable for multiple applications: occluding contours and surface patch orientations provide information about the shape of an object while sharp *albedo* changes are carriers of important texture information.

A variety of different image processing operators can be applied to detect edges, the strength of the response indicates the brightness contrast *i.e.* or the sharpness of the change. As light gets brighter or darker, or any other procedure involved in the image formation changes *e.g*, surface orientation, camera aperture and sensibility, the resulting image gets brighter and the original image I is scaled by some value s as sI, and the magnitude of the gradient passes from $\|\nabla I\|$ to $s\|\nabla I\|$. Under these circumstances gradient strength is affected and undesirable effects for the purpose of edge detection or shape recognition may appear. A simple solution to this problem is to represent gradients by their orientation in the place of their magnitude.

In the rest of this subsection explores some of the most common operators used to compute the spatial and temporal gradients in gray-scale and color images. The theory gray-scale image gradients is pretty well established and some simple operators like the Sobel operator are widely used in many application fields, unfortunately, this is not the case of color images which is a more complicated phenomenon and where a vast gamma of color operators are required to express different color properties.

Gray-scale gradients

For an image I, the gradient operator is expressed as

$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) \tag{1.7}$$

which is estimated by

$$\frac{\partial I}{\partial x} = \lim_{\delta x \to 0} \frac{I(x + \delta x, y) - I(x, y)}{\delta x} \approx I_{i+1,j} - I_{i,j}. \tag{1.8}$$

When images are treated as 2D-discrete functions, their gradients are 2D vectors defined by their horizontal and vertical directional derivatives calculated at each pixel location. The brightness gradient points to the direction of largest possible intensity increase. The strength (magnitude) of the gradient corresponds to the rate of change attained in the gradient vector direction.

If the filters responses are linear, the order in which they are applied is not important (*e.g.*, differentiation after smoothing, or smoothing an image derivative), or integrated into a single operator that applies the smoothing and the differentiation at the same time

$$\frac{\partial (G_\sigma * I)}{\partial x} = \left(\frac{\partial G_\sigma}{\partial x} \right) * I, \tag{1.9}$$

where $\left(\frac{\partial G_\sigma}{\partial x}\right)$ is the derivative of the Gaussian smoothing kernel. Parameter σ controls the scale of the smoothing, and has a substantial effect on the response of a derivative filter (only image artifacts of a size larger than σ will be still distinguishable from the background rendering a recognizable operator response).

The Sobel operator [105] is one of the most popular and most basic differential operators in image processing. The operator convolves two 3×3 kernels with the original image to calculate the directional derivative approximations (one horizontal and one vertical), these kernels already incorporate some smoothing.

Edges are obtained after refining gradient responses using classical image processing techniques such as Canny's method [22] which extracts local maxima in the direction of the gradient vector (thinning the edges to one pixel) and filtering out pixels representing noise by hysteresis thresholding.

Color gradients

In principle, gradients and edge operators for color images (or vectorial images in general) are similar to their gray-scale counterparts, here, trichromatic or spectral images are used to obtain single component images representing gradient's magnitude or edges binary images (indicating if a pixel belongs to an edge or not). Each color channel is processed marginally, but at some point in the processing chain marginal results need to be fused. Different approaches may be followed:

- **Marginal gradients fusion**: The idea behind these methods is to fuse the results obtained from the individual components. Fusion can be done on the individual binary edges maps using logical operators (AND/OR). Gradient magnitude may represent the confidence given to the decision of the pixel belonging to and edge or not. Some methods normalize these magnitudes obtaining values between 0 and 1 and use diffuse logic operators.

- **Method of Di Zenzo**: [35]: Here, color gradient vectors are calculated looking for the direction of maximal change. This is achieved by maximizing the $\mathcal{L}^2$ norm which evaluates the vectorial distance at the colorimetric space.

- **Hue based methods**: This approach for fusing marginal gradients is exclusive of the Hue-Saturation-Value (***HSV***) color space. The principle is that highly saturated colors are insensible to noise and deserve higher confidence than low saturated ones. Carron and Lambert's method [24] weights the individual gradients and mixes them according to their saturation. Using one of the following alternatives:

$$\begin{aligned} \mathbf{G}_{\text{Carron 1}} &= \alpha\mathbf{G}^H + \mathbf{G}^S + \mathbf{G}^V, \\ \mathbf{G}_{\text{Carron 2}} &= \alpha\mathbf{G}^H + (1-\alpha)\mathbf{G}^S + (1-\alpha)\mathbf{G}^V, \end{aligned} \tag{1.10}$$

  where α is a function whose value depends on the saturation $\alpha = \alpha(S)$ (indicating the relevance of the hue), $\mathbf{G}^H$, $\mathbf{G}^V$ and $\mathbf{G}^S$ are the individual hue, saturation and value gradients. Some precautions are needed to apply this method correctly: hue is a circular type information and the relevance accorded to it must consider the saturation of all the pixels involved during the gradient computation (neighborhood's geometrical mean is proposed to this purpose).

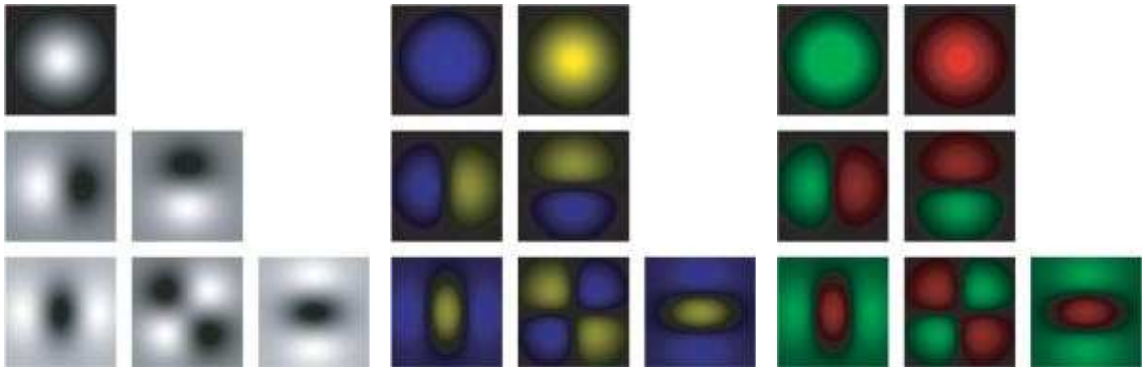


Figure 1-5: Gaussian smoothing and derivative filters in gray-scale and color up to second order in $\mathbf{x} = (x, y)$(spatial differentiation) and λ (spectral differentiation). Image taken from [21].

- **Gaussian color model derivatives**: Gaussian models are used too to define the spectral derivatives. Supposing a Gaussian centered at a fixed wavelength with a fixed spectral bandwidth (standard deviation) there are three spectral derivatives available: E which is the zero-order derivative, the first order derivative E_λ (which compares the yellow and the blue parts of the spectrum), and the second-order derivative $E_{\lambda\lambda}$ that compares the green middle part of the spectrum to the outer regions on it (these parameters are implemented on common camera devices imitating the properties of human vision). Convolving the spectral Gaussian measurements $E, E_\lambda, E_{\lambda\lambda}$ with a Gaussian derivative kernel the spatio-spectral derivatives of a color image are estimated (spatial position and scale are controlled by the Gaussian derivative parameters). The total edge strength attributed to color information is expressed as

$$E_W = \sqrt{E_x^2 + E_y^2 + E_{\lambda x}^2 + E_{\lambda y}^2 + E_{\lambda\lambda x}^2 + E_{\lambda\lambda y}^2}, \tag{1.11}$$

  the spatio-spectral exploration (color receptive fields) up to the second order derivatives is depicted in **Figure 1-5**.

As for gray-scale images, marginal gradients are obtained after applying a differential filter/operator such as Sobel or Prewitt to each individual channel, but color image methods must to avoid the introduction of false chromaticities. In addition, color differential must combine separate channel derivatives without loss of information.

Other image derivative operators

Second order derivatives such as the Laplacian filters, are used to find areas of rapid change (edges) in images. The ***Laplacian*** operator is defined as

$$L(x, y) = \nabla^2 I(x, y) = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial I(x, y)}{\partial y^2}. \tag{1.12}$$

It is known that derivative filters response can vary significantly if pixels are affected by noise (*i.e.* derivative operators are sensitive to noise), and a common practice is to smooth the image using a low pass filter such as the ***Gaussian*** filter

$$g(x, y; \sigma) = \frac{1}{2\pi\sigma} e^{\frac{-(x^2+y^2)}{2\sigma}}, \tag{1.13}$$

before applying the Laplacian (σ is a parameter which controls the scale of the smoothing kernel). The complete process is thus called ***Laplacian of Gaussian*** and it is abbreviated as the ***LoG*** operator

$$L(x, y; \sigma) = \nabla^2 g(x, y; \sigma) * I(x, y). \tag{1.14}$$

Other popular operator is the ***Difference of Gaussians*** operator or ***DoG***, it represents a feature enhancement technique which involves calculating the difference between two smoothed images blurred with two different Gaussian kernels (the value of the scale parameter σ is different). As Gaussian kernels suppress high-frequency spatial information the subtraction of two smoothed images preserves only the spatial information that lies inside the range of frequencies that are preserved by both kernels. The ***DoG*** is a band-pass filter that discards and selects just a set of spatial frequencies of the original grayscale image. Mathematically, the ***DoG*** is expressed as

$$\Gamma_{\sigma, K\sigma}(x, y) = I(x, y) * \frac{1}{2\pi\sigma} e^{\frac{-(x^2+y^2)}{2\sigma}} - \frac{1}{2\pi K^2\sigma} e^{\frac{-(x^2+y^2)}{2K^2\sigma}}. \tag{1.15}$$

As shown by Lindeberg in [85] the ***DoG*** operator is equivalent to the ***LoG*** and it is faster to compute. This property is be very useful to compute the Scale Invariant Feature Transform (SIFT) (see subsection 1.1.4 in page 40).

1.1.3 Texture analysis

Texture is a phenomenon which depends on the scale at which different objects or artifacts appear in an image. A large collection of small objects is best thought of as a texture. Textures appear as repetitions of a single local patch. Though, each repetition may be distorted by a viewing transformation. Texture analysis is widely used in computer vision and computer graphics. In computer vision, they provide a very strong cue to object identity and the possible analysis of material properties (*e.g.* hardness, smoothness, opaqueness). Different representations of textures exist depending on the problem. *Local texture* representations encode the texture in the local neighborhood of a point in the image, this kind of encoding is useful for image segmentation and object recognition. *Pooled* representations are used to determine what texture is represented by a patch on an image and it is employed for material recognition applications. Finally, *data-driven* texture representations are used to model a texture and generate (synthesize) a textured region, it is useful for computer graphics applications to draw in image missing regions (filling holes) so that they look natural, this method is popularly known as *inpainting*. Finally, the deformation of the texture on a surface allows us to perform inference about the geometrical transformations parameters, or the local curvatures at different points on the surface. These methods are known as *shape from texture*.

All the existing techniques can be grouped into four main categories: statistical, geometrical, model-based and signal-processing methods. Statistical methods analyze first order gray level statistics based on histograms of local differences, second order statistics can be on the form of co-occurrence matrices. Then, signal processing methods are based on bank of filters such as multichannel Gabor filtering or wavelets. Finally, model-based methods are based on Markov random fields or fractals.

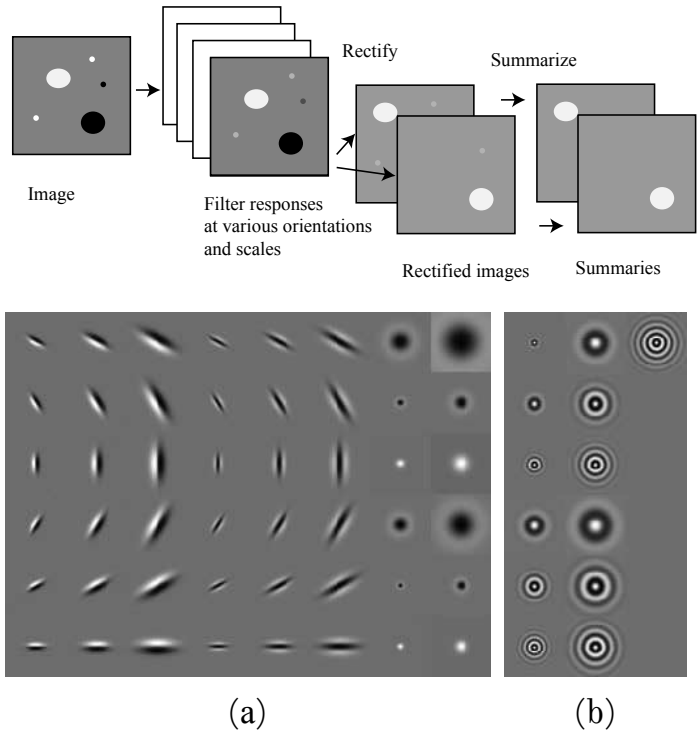


Figure 1-6: The scheme on the first row show how to build local texture representations by filtering an image with a set of filters of various scales and summarizing their individual responses. An example of a possible set of filters is shown on the second row: (a) 48 oriented filters, and (b) a set of orientation invariant filters. Image taken from [45].

Filter banks

Texture can be modeled as the repetition of certain primitive elements *e.g.* spots, bars, edges, sometimes referred as *textons*. A natural representation of a texture is to describe what *textons* the texture exhibits and how these elements repeat. Some special types of filters are required to denote the presence of such primitive shapes. Each point in the image is represented by the set of elements found nearby. As the whole texture repeats forming a pattern, primitive elements are repeated as well. Each filter acts as a sub-element detector at a collection of scales and orientations and each point is mapped to a vector of filter responses calculated at that point. In general, this vector gives a description about the similarity of the neighborhood around our pixel in comparison to each *texton* sub-element in a diversity of scales and orientations. Orientation invariant filters can be used as well as an alternative to the use of multiple oriented filters [42]. Conforming a complex pattern of concentric spots (see **Figure 1-6-(b)**).

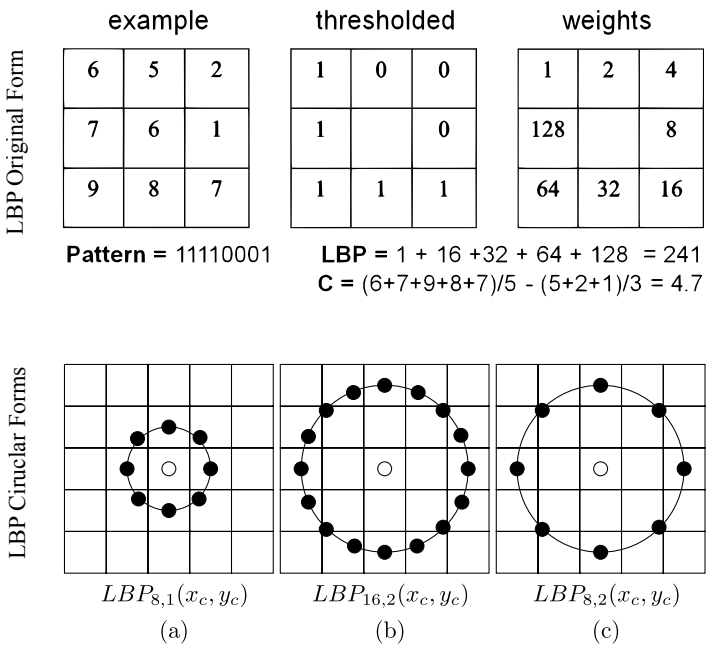


Figure 1-7: An example of the *LBP* original forms appears in the first row: all pixels in the neighborhood are thresholded by the pixel at the center, then a binary code is formed by the dot product of the thresholded values and their weights. Different sized neighborhoods of the form $LBP_{P,R}(x_c, y_c)$
appear on the second row. Here P is the number of pixels considered by *LBP* pattern and R the radius distance from the neighborhood center [107].

Local binary patterns

Signal processing methods based on filter banks are computationally too complex to meet the real-time requirements of many computer vision applications. Recently, more economical and still discriminative methods such as local binary patterns (*LBP*) have been proposed to deal with this problem. Local binary patterns are very efficient texture operators which label the neighborhood of each pixel as a binary number according to a predefined set of comparisons. One important advantage of *LBP* operators is that they are invariant against monotonic gray level changes caused *e.g.* by illumination variations. *LBP* operators are computationally simple and very discriminant, due to these attributes they have a considerable success, showing that filter banks with large support areas are not always necessary to achieve high performance in texture classification. The original *LBP* pattern operator was introduced by Ojala *et al*. in [101] where the authors highlighted two important aspects of a texture: a pattern and its strength. The original and most basic operator consists of a 3×3 square neighborhood where its center is used as a threshold, the resulting thresholded values are accumulated considering a weight associated to their position, these weights grow as powers of two: $\{2^0, 2^1, \cdots, 2^{P-1}\}$. The first row in **Figure 1-7** illustrates the operator concept, the idea is to build a binary code of length P (*i.e.*, for a 8-pixel neighborhood there are $2^8 = 256$ possible labels) which depends on the relative gray values of the center and the pixels in the neighborhood. Several years after the original publication, this basic operator was presented in a more generic form by Ojala *et al*. [102]. In its generic form, the operator has no limitations on the neighborhood's size or on the number of sampling points (P).

The local texture of the monochromatic image $I(x, y)$ is characterized by the joint distribution of $P + 1$ gray values

$$T = t(g_c, g_0, g_1, \cdots, g_{P-1}) \tag{1.16}$$

where g_c is the gray value of an arbitrary pixel (x, y) (*i.e.*, $g_c = I(x, y)$), and $g_p = I(x_p, y_p)$ is the gray value of a sampling point $p \in \{0, \cdots, P - 1\}$ located at

$$\begin{aligned} x_p = &\quad x + Rcos(2\pi p/P) \\ y_p = &\quad y - Rsin(2\pi p/P) \end{aligned} \tag{1.17}$$

where P is the total number of sampling pixels around (x_c, y_c), these samples are taken sampling the circle of radius R at evenly spaced degrees. The T distribution is approximated by the local neighborhood differences

$$T \approx t(g_0 - g_c, g_1 - g_c, \cdots, g_{P-1} - g_c), \tag{1.18}$$

this makes the operator invariant to changes of the mean gray value. But these values are not invariant to other changes in gray levels, to alleviate this limitation only signs of the differences are considered

$$t(s(g_0 - g_c), s(g_1 - g_c), \cdots, s(g_{P-1} - g_c)) \tag{1.19}$$

where $s(z)$ is the step function

$$s(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}. \tag{1.20}$$

Accumulating the thresholded differences weighted by powers of two, the operator is finally defined as

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c)2^p. \tag{1.21}$$

Equation (1.21) interprets the sign of the differences in a neighborhood as a P-bit binary number, which results in 2^P different labels. The local gray-scale texture is approximately described with the 2^P-bin distribution of *LBP* labels

$$T \approx t(LBP_{P,R}(x_c, y_c)). \tag{1.22}$$

Pixels located at the diagonals are interpolated. While this may seem costly, this generic form is advantageous as it is adaptable for different scales sizes, and rotation invariant descriptors are easily derived from this circular neighborhood indexing.

Uniform patterns (*ULBP*) [102] are the result of another modifications made to the original definition of the *LBP* operator. Here, a uniformity measure that counts the number of bitwise transitions from 0 to 1 or vice versa is employed. A pattern is considered uniform if its uniformity measure is ≤ 2. For example, 00000000 and 01110000 are uniform patterns but 11001001 and 10010011 are not. It has been proven experimentally that *ULBP* patterns account for the majority of

texture patterns in natural images, and they are indications that uniform patterns are more stable or less prone to noise. *ULBP* patterns are also convenient because they reduce the possible number of labels conducting to more concentrated (less sparse) histogram distributions.

An excellent source of information about the *LBP* operators and its applications in computer vision is the book of Pietikäinen [108]. In **Section 4.1.2** it is described how to use *ULBP*'s to estimate the angles in a circular pattern and how this information is embedded inside covariance matrices for texture and gesture recognition and object tracking applications.

1.1.4 Local features (interest points)

Local feature detection and feature description are two linked but individual processes. In many computer vision applications, first a detector is applied to locate the features (*e.g.* edges, corners, t-junctions), and after selecting the most important features their descriptors are then constructed to represent them.

Many feature based techniques have been used since the early days of computer vision being of particular interest the works of Hannah [60], Moravec [97] and Harris and Stephens [61]. There are two main approaches for finding feature points correspondences: 1) searching locally for the highest correlation location (or the least square errors sum) or 2) independently detecting features in all the images and match based on their local appearance. Local search is preferred when images are taken in nearby points or in a rapid succession (high frame video sequences). On the other hand, matching is preferred for larger motions or long term occlusions.

Local invariant features provide representations that allow to efficiently match local structures between images. Approximately the same set of local features needs to be extracted from two images showing the same object: local features must be located precisely and in a repeatable fashion. A second criteria to meet is that features should be *distinctive*, allowing the algorithms to identify them in a list. To resist occlusions, a sufficient number of features evenly distributed along the target is typically demanded.

Corner and feature point detectors

A procedure to find the set of key-points that are stable and easy to identify is the initial step for the local feature representation. In the recent years computer vision has seen an explosion on the number of corner of feature points detectors and descriptors: Harris [124], SIFT [89], SURF [10] and FAST [119], etc. To keep brevity, only the most popular ones will be described here: Harris and SIFT.

Harris detection method was designed to find features that are easy to track. They showed how textureless patches patches are nearly impossible to localize, contrary to patches with great contrast changes. Gradients in a patch are not the only condition required, straight lines (ensemble of locally connected points sharing the same gradient orientation) suffer from the aperture problem (see [90] and [66]) which says that: *"it is only possible to align features along the direction normal to the edge direction"*. Locations with at least two directions are easier to localize than straight lines. The examples contained in **Figure 1-9** exemplify this problem.

Comparing one image patch against itself while introducing a very small variation in the position $\Delta\mathbf{u}$ provides an

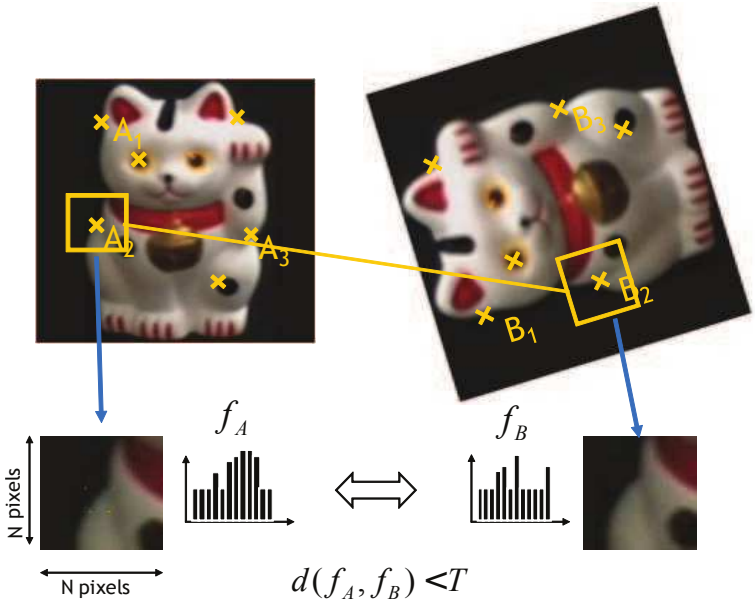


Figure 1-8: Object recognition with local features requires the algorithm to extract the sets of key-points in both images. For each key-point, the surrounding region is normalized for rotation and scale changes. Local descriptors are built then for each normalized region. Finally, feature matching is performed by comparing the local descriptors using an appropriate similarity measure. Image taken from [130].

estimation about the stability of the feature. This is the *autocorrelation* function of a surface patch

$$E_{AC}(\Delta\mathbf{u}) = \sum_i w(\mathbf{x}_i) \left[I_0(\mathbf{x}_i + \Delta\mathbf{u}) - I_0(\mathbf{x}_i) \right], \tag{1.23}$$

where I_0 and I_1 are just two images being compared and $\mathbf{u} = (u, v)$ is the displacement vector between the two, $w(\mathbf{x}_i)$ is an spatially varying weighting function such as the Gaussian kernel. The summation is done here for every pixel $\mathbf{x}_i$ inside the patch, pixels are indexed by i.

When a patch has a compactly localized and strong minimum it means it is stable and easy to find, edges following a straight line can be ambiguous in one direction while smooth textureless patches are nearly impossible to localize because they have no stable minimum.

This concept was formalized by Lucas *et al*. and Shi and Tomasi in [90] and [124] respectively. The autocorrelation surface can be approximated by the truncated Taylor series: $I_0(\mathbf{x}_i + \Delta\mathbf{u}) \approx I_0(\mathbf{x}_i) + \nabla I_0(\mathbf{x}_i) \cdot \Delta\mathbf{u}$. Substituting this into equation (1.23) we have

$$E_{AC}(\Delta\mathbf{u}) \approx \Delta\mathbf{u}^T \mathbf{A} \Delta\mathbf{u} \tag{1.24}$$

where $\nabla I_0(\mathbf{x}_i) = \left(\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y} \right)$ and $\mathbf{A}$ is the weighted autocorrelation matrix

$$\mathbf{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}. \tag{1.25}$$

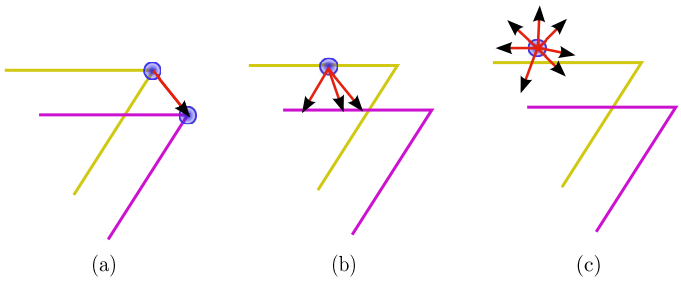


Figure 1-9: The aperture problem is illustrated with three relevant cases: (a) a corner-like point is given and a unique and stable correspondence is relatively easy to find, (b) the point is located over an edge, as many points can be positioned over and edge the resulting flow is not unique and poorly defined, (c) a textureless point is given, even more candidate displacements exists and the point could move in any direction.

Matrix **A** must be interpreted as a tensor (a multi-band image), the authors applied eigenvalue analysis to it getting two eigenvalues (λ_0, λ_1) and two eigenvector directions. They observed that the degree of uncertainty for the key-point autocorrelation energy concentration depends mostly in the smaller of the eigenvalues $min(\lambda_0, \lambda_1)$, so good features to track are those where the minimum eigenvalue is above a certain threshold.

Förstner [43] and Harris and Stephens [61] improved this key-point location criteria proposing a rotationally invariant scalar operator applied to weighted autocorrelation matrix **A**. The method of Harris and Stephens avoids the costly eigenvalue analysis that requires the calculation of square roots using the following mathematical property

$$\det(\mathbf{A}) - \alpha \operatorname{trace}(\mathbf{A})^2 = \lambda_0\lambda_1 - \alpha(\lambda_0 + \lambda_1)^2. \tag{1.26}$$

In practical terms any coarsity measure which downweights the value of edge-like features where $\lambda_{max} \gg \lambda_{min}$ is useful. The typical steps towards feature points detection outlined in **Algorithm 5**.

Algorithm 5: Basic feature point detection algorithm

Data: Input image I

Result: Coarsity image K

1 The input image I is convolved with the derivatives of a Gaussian to get the horizontal and vertical derivatives I_x and I_y.

2 From I_x and I_y three new images are computed corresponding to the outer products of these gradients: I_x^2, I_xI_y and I_y^2.

3 Images I_x^2, I_xI_y and I_y^2 are convolved with a larger Gaussian (the resulting images define the autocorrelation matrix **A** for each location).

4 At each location, compute any of the scalar coarsity (interest) measures (*e.g.* Harris corners (1.26)).

Schmid *et al*. performed tests to evaluate many of the different interest point detectors that have been proposed in computer vision [120]. They first evaluated the detection repeatability, that is how many times a feature is kept under varying conditions such as rotations, scale and illumination changes and the presence of noise. Secondly they evaluated the *information* content available at each of the detected feature points. They defined the information content as the entropy

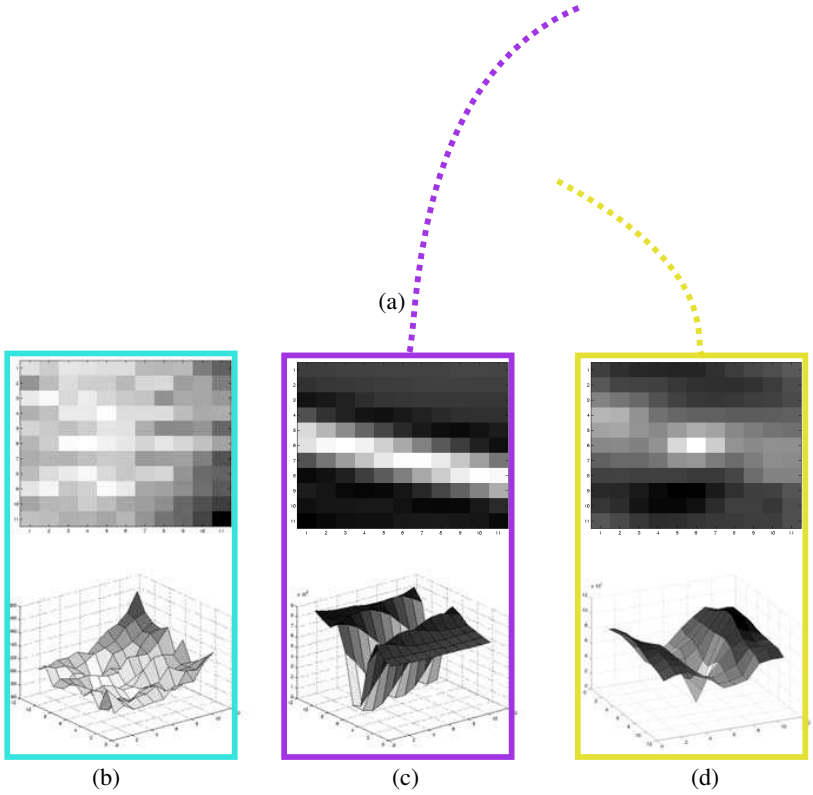


Figure 1-10: Autocorrelation function for three different types of patches take from the image in (a). Patch in (b) is taken from the cloud, no distinguishable peaks are found in its autocorrelation neighborhood, the patch in (c) comes from the roof edge, this patch suffers from the 1D aperture problem (path location can be confused along the edge) and finally, patch in (d) comes from the flower bed and a good unique minimum of the auto-correlation function allows easy localization. Figures taken from [130]

.

of a set of rotationally invariant grayscale descriptors. They found that the smoothed (Gaussian derivative) version of the Harris operator using parameters $\sigma_d = 1$ (Gaussian derivative scale) and $\sigma_i = 2$ (scale of the smoothing of $\mathbf{A}$) outperforms the others.

Scale invariant feature transform (*SIFT*)

In images containing low-frequency detail, fine scale-features may not exist, but when this images are regarded with a lower level of detail (at a lower resolution) some feature points begin to appear. Considering this problem some methods extract features with a variety of scales *e.g.*, using a pyramid of images and then matching features belonging to the same pyramid level. This approach works fine for comparing images taken in similar conditions and without significant scale changes, however, in applications such as object tracking or recognition, the scale of the object in the image is unknown and can be drastically different from one sample to the other.

A different and more applicable approach is to detect features which are invariant to scale changes. Mikolajczyk and

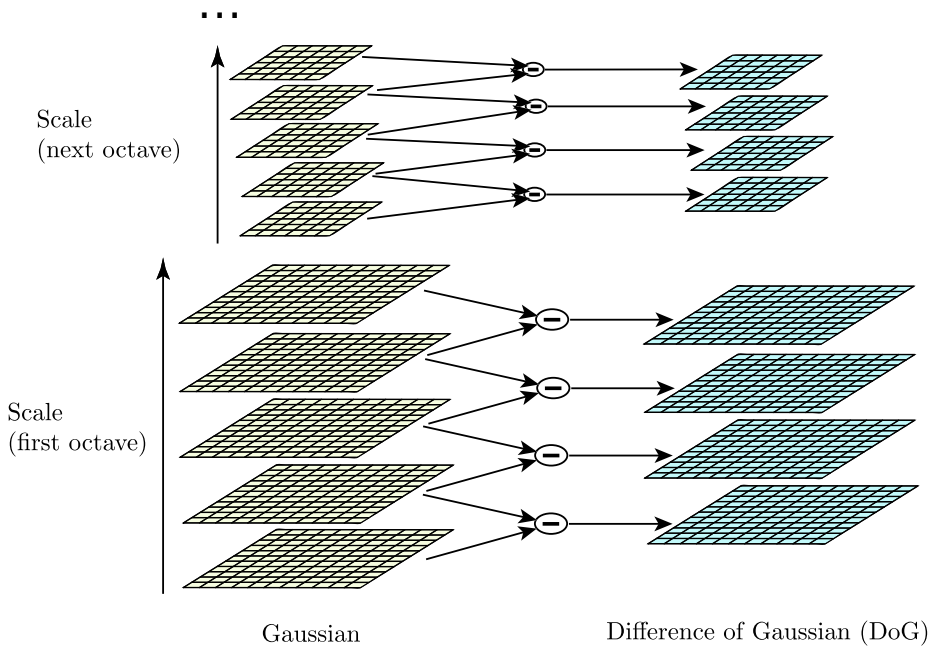


Figure 1-11: Stack of smoothed images $L(x, y, \sigma)$ calculated by *SIFT* and the *DoG* computed from them. Image taken from [89].

Schmid [94] and Lowe [89] developed feature detectors following this idea and obtained in result two different methods based on different assumptions.

The works of Lindeberg are considered within the most relevant in computer vision because he established the theory that formally analyzes the effects of scale. In his works [84] and [85] he helped to define the concept of *characteristic scale* for automatic scale selection using the extrema of the Laplacian of Gaussian (*LoG*) function.

Based on these works, Lowe developed the Scale Invariant Feature Transform (*SIFT*) [89] which constructs a space-scale $L(x, y, \sigma)$ with the repeated convolution of a variable-scale Gaussian $G(x, y, \sigma)$ with the input image $I(x, y)$.

Figure 1-11 shows how *SIFT* starts by incrementally convolving the image with a set of Gaussian kernels $G(x, y, \sigma)$, that produces an scale-space stack of images (left-column). Each image in the stack is separated by a factor k in scale-space. *SIFT* separates this convolutions into octaves (doublings of σ), each octave is subdivided into an integer number of intervals s, then the value of k is $k = 2^{1/s}$. For extrema detection, it is necessary to produce $s + 3$ images in the stack of blurred images for each octave. When a complete octave has been processed, the Gaussian image that has twice the initial value of σ is re-sampled by taking every second pixel in each row and column. The stack of *DoG* (**Figure 1-11** second column) images results from the subtraction of adjacent images in the scale-space representation as defined by equation (1.15).

From the stacks of *DoG* images local extrema are detected comparing each sample point to its eight neighbors in the current image and width the nine neighbors in the scale above and below. Only values which are larger than all their neighbors or smaller than all of them are selected. The cost of this is not that high because most of the candidate points are discarded with just a few comparisons. Because point extrema can be arbitrary close together, there is no minimum spacing of samples that warranties the detection of all extrema. The solution is a trade off between efficiency and completeness, experiments show that many of the extrema that are close together are quite unstable and disappear with small image

perturbations.

As the detection of *SIFT* points is based on a different concept than the autocorrelation stability of Harris-like operators these methods complement each other and can be used together. To robustify the Harris operator, Mikolajczyk and Schmid introduced in [94] a scale selection mechanism evaluating the *LoG* function at each one of the Harris points found in a multi-scale pyramid and to keep the extreme points only. The complete algorithm for the detection of *SIFT* points is summarized in **Algorithm 6**.

Algorithm 6: Scale Invariant Feature Transform (*SIFT*) point detection algorithm

Data: Input image I

Result: List of *SIFT* points p

1 For each octave, construct a scale-space stack of $s+3$ blurred images $L(x, y, \sigma)$ by applying the Gaussian kernel $G(x, y, \sigma)$ where $\sigma \in \{\sigma_0, k\sigma_0, k^2\sigma_0, \cdots, 2\sigma_0\}$ and $k = 2^{1/s}$.

2 For each octave stack obtain the stack of differentials of Gaussians (*DoG*) doing the subtraction $D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$.

3 Compare each point in $D(x, y, \sigma)$ to its 26 scale-space neighbors and keep insert in a list p those which are larger than all or smaller than all.

4 For each point $x \in p$ estimate the precise sub-pixel and sub-scale offset $\hat{\mathbf{x}}$ by quadratic fitting: $\hat{\mathbf{x}} = \frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}$ and update $\mathbf{x}$ as $\mathbf{x} \longleftarrow \mathbf{x} + \hat{\mathbf{x}}$.

5 For each point $\mathbf{x} \in p$ calculate $D(\mathbf{x})$ and remove from p if $|D(\mathbf{x})| < D_{\text{thresh}}$.

6 For each point $\mathbf{x} \in p$ calculate the Hessian matrix $\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$ and remove from p if $\frac{\text{trace}(\mathbf{H})^2}{\det(\mathbf{H})} > \frac{(r+1)^2}{r}$ where $r = 10$.

SIFT algorithm is for sure one of the most transcendental algorithms in computer vision in the recent years. Because its robustness, it represents the milestone for many other applications such as 3D object reconstruction and object retrieval. It biggest problem however is the execution speed, focusing on this problem Bay *et al*. published in [10] their *Speeded Up Robust Features* (*SURF*), an alternative method which is several times faster than *SIFT* and claimed by their authors to be more robust against different image transformations. *SURF* is based on Haar wavelets and integral images some concepts that we will discuss in **Chapter 2** when describing Viola and Jones algorithm used for the detection of faces on images and videos.

1.1.5 Motion information

The importance of movement in computer vision cannot be overestimated. It is used in a broad range of applications such as: structure from motion, camera stabilization, object detection by background subtraction and optical flow just to name a few. Background subtraction and optical flow are for sure the more relevant methods in the context of object detection and tracking applications, so our discussion about motion information in computer vision will concentrate on this methods.

Background subtraction is a technique to extract the foreground of an image yielding regions and pixels of interest for the localization of moving objects. Unfortunately, the detection of moving objects with these techniques is somehow

limited to static camera configurations. A background images is compared against the current image to detect moving pixels and regions, these background images can simply be the preceding image on the sequence or a more elaborate model that incorporates more information from the past.

Optical flow generally involves the minimization of the brightness or color difference between correspondent pixels. Lucas *et al*. [90] *KLT* optical flow algorithm computes independent estimates of the motion of each pixel, uses patch-based approach to estimate local image displacements with sub-pixel precision using a Taylor series expansion.

1.1.5.1 Background subtraction methods

Two relevant techniques are discussed here: the $\Sigma\Delta$ background subtraction algorithm presented by Lacassagne *et al*. [77] and the multi-layer algorithm based on color and texture of Yao and Odobez [152]. The first method is super fast and yields very acceptable results while the second one is slower but gives finer results.

Background subtraction is a widely used approach for detecting moving objects in videos captured from static cameras. The basic principle is that as objects move they affect the image formation process and pixels related to the objects location in the image change more than those who belong to the static background. Differentiating pixel-by-pixel the current frame and a reference frame in principle should be enough. The absolute frame difference at time t, noted D_t shows higher intensities for the pixel locations $\mathbf{x} = (x, y)$ which changed in the two frames. The problem with this approach is that it only works for the cases where all foreground pixels are moving and all the rest remain static. The true is that the background is never completely static and lighting changes, waving trees, clouds, shadows and complex phenomena related to the image formation process produce small or even important changes in brightness, texture and color. To reduce the number of false moving pixels detected a threshold can be helpful to filter out insignificant motion but it is never easy to propose a threshold that works for all the cases. A different possibility is to construct a reference background image by computing the average of a series of preceding images (containing only background is naturally preferred). **Figure 1-12** illustrates how foreground objects are detected using background subtraction.

Wren *et al*. propose in [148] to fit a Gaussian probability density function for the previous n pixels. A running average algorithm is an efficient algorithm that avoids recalculating averages for complete at each iteration

$$\mu_t(\mathbf{x}) = \alpha I_t(\mathbf{x}) + (1 - \alpha)\mu_{t-1}(\mathbf{x}) \tag{1.27}$$

where $I_t(\mathbf{x})$ is the pixel's $\mathbf{x}$ current value and $\mu_t(\mathbf{x})$ its previous average. Parameter $\alpha \in \mathbb{R}[0, 1]$ controls the linear combination in (1.27) and defines the flexibility of the background model update (when $\alpha \to 1$ pixel's model is updated faster and when $\alpha \to 0$ it is more conservative). The standard deviation $\sigma_t(\mathbf{x})$ of the Gaussian is calculated in a similar fashion as $\mu_t(\mathbf{x})$, then a pixel is labeled as foreground if the following inequality holds

$$|I_t(\mathbf{x}) - \mu_t(\mathbf{x})| > k\sigma_t(\mathbf{x}). \tag{1.28}$$

Stauffer and Grimson assume that a single Gaussian is not enough for modeling the pixels behavior and propose a Gaussian mixture model (a weighted sum of multiple Gaussian models) as a solution [126].

a) **$\Sigma\Delta$ background subtraction**: This background subtraction method estimates the parameters of the background using

$\Sigma\Delta$ modulation which is a very popular tool in analog-to-digital conversion. This method requires very few resources in comparison to other background subtraction methods which makes it attractive for real-time implementations in embedded processors. Let f_t be a time-varying continuous or discrete 1D signal, a discrete signal d_t is estimated at discrete time indexes $\{t_i\}_{i\in\mathbb{N}}$, and at every discrete value t_i the following update formulas are executed

$$d_{t_i} = \begin{cases} d_{t_i} - \varepsilon, & \text{if } d_{t_{i-1}} < f_{t_i} \\ d_{t_i} + \varepsilon, & \text{if } d_{t_{i-1}} \geq f_{t_i}. \end{cases} \tag{1.29}$$

When applied for background subtraction, the input signal is now the value at pixel location $\mathbf{x} = (x, y)$ over time $I_t(\mathbf{x})$. A first order $\Sigma\Delta$ background $M_t(\mathbf{x})$ estimator is computed from it. Then, the values of the absolute differences $|I_t(\mathbf{x}) - M_t(\mathbf{x})|$ are used to compute a second order $\Sigma\Delta$ background estimator $V_t(\mathbf{x})$ which is considered a parameter of dispersion. The basic version of the $\Sigma\Delta$ background subtraction is shown in **Algorithm 7**, where the $\Sigma\Delta$ background model M_t and its $\Sigma\Delta$ variance V_t are updated every frame. N is an amplification factor for V_t. Motion is detected for every pixel $\mathbf{x}$ where $O_t(\mathbf{x}) \geq V_t(\mathbf{x})$. V_{min} and V_{max} are two parameters used to control possible overflows in $V_t(\mathbf{x})$. This basic algorithm has been enhanced to provide cleaner results using morphology operators and other modifications, more details are found in [77] and [92].

Algorithm 7: $\Sigma\Delta$ background subtraction algorithm.

Data: Input image I_t
Result: Binary image $\hat{E}_t$

foreach *pixel* $\boldsymbol{x}$ **do**
  if $M_{t-1}(\mathbf{x}) < I_t(\mathbf{x})$ **then** $M_t(\mathbf{x}) \leftarrow M_{t-1}(\mathbf{x}) + 1$
  if $M_{t-1}(\mathbf{x}) > I_t(\mathbf{x})$ **then** $M_t(\mathbf{x}) \leftarrow M_{t-1}(\mathbf{x}) - 1$
  otherwise $M_t(\mathbf{x}) \leftarrow M_{t-1}(\mathbf{x})$
foreach *pixel* $\boldsymbol{x}$ **do**
  $O_t(\mathbf{x}) = |M_t(\mathbf{x}) - I_t(\mathbf{x})|$
foreach *pixel* $\boldsymbol{x}$ **do**
  if $V_{t-1}(\mathbf{x}) < N \times O_t(\mathbf{x})$ **then** $V_t(\mathbf{x}) \leftarrow V_{t-1}(\mathbf{x}) + 1$
  if $V_{t-1}(\mathbf{x}) > N \times O_t(\mathbf{x})$ **then** $V_t(\mathbf{x}) \leftarrow V_{t-1}(\mathbf{x}) - 1$
  otherwise $V_t(\mathbf{x}) \leftarrow \max(\min(V_t(\mathbf{x}), V_{\text{max}}), V_{\text{min}})$
foreach *pixel* $\boldsymbol{x}$ **do**
  if $O_t(\mathbf{x}) < V_t(\mathbf{x})$ **then** $\hat{E}_t(\mathbf{x}) \leftarrow 0$ **else** $\hat{E}_t(\mathbf{x}) \leftarrow 1$

b) **Multi-Layer Background Subtraction Based on Color and Texture**: Yao and Odobez propose in [152] a multilayer background subtraction technique which takes the advantages of local texture features represented *LBP*'s and photometric invariant color measurements in the *RGB* color space. The idea is to complement the advantages of each type of measurement. On its hand, *LBP*'s are very stable and robust to light variations in rich texture regions but weak on uniform regions. Color information overcomes *LBP*'s limitation on those regions. Both the *LBP* feature and the selected color feature are able to handle local illumination changes such as shadows from moving objects.

Let $\mathcal{I} = \{I_t(\mathbf{x})\}_{t=1,\ldots,N}$ be an image sequence acquired with a static camera. The background model learned up to time t is denoted as $\mathcal{M} = \{M_t(\mathbf{x})\}$. At each time t the model $M_t(\mathbf{x})$ is a structure containing $M_t(\mathbf{x}) = \{K_t(\mathbf{x}), \{\mathbf{m}_{t,k}(\mathbf{x})\}, B_t(\mathbf{x})\}$, which consists of a list of $K_t(\mathbf{x})$ modes $\{\mathbf{m}_{t,k}(\mathbf{x})\}_{k=1,\cdots,K_t(\mathbf{x})}$ where the first $B^t(\mathbf{x}) \leq K_t(\mathbf{x})$ have been identified as

background pixels (at different layers possibly). There is a limit for the number of modes a pixel $\mathbf{x}$ can store which is $K_{\max}$. Each mode $\mathbf{m}_{t,k}(\mathbf{x})$ consists of 7 components $\mathbf{m}_k = \{I_k, \hat{I}_k, \check{I}_k, LBP_k, w_k, \hat{w}_k, L_k\}$:

- I_k is the average *RGB* vector (I_k^R, I_k^G, I_k^B) of the mode k.
- $\hat{I}_k$ and $\check{I}_k$ denote the estimated minimal and maximal *RGB* vectors (maximums and minimums of each R,G and B channels are calculated independently) associated to the mode k.
- LBP_k is the average *LBP* pattern from pixels assigned to this mode k (average of the 1's or 0's of the pattern are calculated independently).
- $w_k \in [0, 1]$ denotes the weight assigned to this mode (*i.e.*, the probability that this mode belongs to the background).
- $\hat{w}_k$ the maximal value that w_k has achieved in the past.
- L_k the background model layer to which this model belongs. $L_k = 0$ means that m_k is not a reliable background mode and $L_k = \ell > 0$ indicates a reliable mode in the ℓ-th layer.

Multi-layered background modeling is able to detect foreground objects against all backgrounds learned from the past but covered by long-term stationary objects (stationary occlusions). An approach which is useful to detect abandoned luggage and background scene changes (*e.g.*, graffiti paintings or posters).

At each frame t the algorithm receives a $LBP_t(\mathbf{x})$ and a *RGB* value $I_t(\mathbf{x})$ measured at the image location $\mathbf{x} = (x, y)$, the algorithm seeking the mode of the background that best match these measurements. The mode which is the closest one to the measurements is $\tilde{k} = \arg\min_k D(\mathbf{m}_{t-1,k})$. If this distance is above a certain threshold (*i.e.*, $D(\mathbf{m}_{t-1,k}) > T_{bg}$), a new mode is created with the parameters $\{I_t, LBP_t, w_{\text{start}}, w_{\text{start}}, 0\}$. Here, w_{start} is the initial weight (confidence in a newer model being in the background should be low). New models are added to the list of modes kept for each pixel $\mathbf{x}$ if $K_{t-1}(\mathbf{x}) < K_{\max}$, if not, it replaces the existent mode with lowest weight. In the case the measurements closely match one of the existent modes, its representation needs to be updated as

$$\begin{aligned}
\check{I}_{t,\tilde{k}} &= \min(I_t, (1+\beta)\check{I}_{t-1,\tilde{k}}) \\
\hat{I}_{t,\tilde{k}} &= \max(I_t, (1-\beta)\hat{I}_{t-1,\tilde{k}}) \\
I_{t,\tilde{k}} &= (1-\alpha)I_{t-1,\tilde{k}} + \alpha I_t \\
LBP_{t,\tilde{k}} &= (1-\alpha)LBP_{t-1,\tilde{k}} + \alpha LBP_t \\
w_{t,\tilde{k}} &= (1-\alpha_w^i)w_{t-1,\tilde{k}} + \alpha_w^i \\
&\quad \text{where } \alpha_w^i = \alpha_w(1+\tau\hat{w}_{t-1,\tilde{k}}) \\
\hat{w}_{t,\tilde{k}} &= \max(\hat{w}_{t-1,\tilde{k}}, w_{t,\tilde{k}}) \\
L_{t,\tilde{k}} &= 1 + \max\{L_{t-1,k}\}_{k=1\cdots K_{t-1}, k\neq\tilde{k}} \\
&\quad \text{if } L_{t,k} = 0 \text{ and } \hat{w}_{t,\tilde{k}} > T_{\text{bw}},
\end{aligned} \tag{1.30}$$

all the other modes are just recopied from the previous frame (*i.e.*, $\mathbf{m}_{t,k} = \mathbf{m}_{t-1,k}$), with the exceptions of the weights which are updated (decreasingly) as

$$w_{t,k} = (1-\alpha_w^d)w_{t-1,k} \text{ with } \alpha_w^d = \frac{\alpha_w}{1+\tau\hat{w}_{t-1,k}}. \tag{1.31}$$

Figure 1-12: Background subtraction methods detect foreground objects by comparing the current image with the background models. The original image is the top-left image, current background representation is shown in top-right image, in the bottom-left we have the distance between the current image and the background image and in the bottom right the foreground image is extracted. Images taken from [152].

Many parameters intervene in the equations above, $\beta \in [0, 1]$ fixes the learning rate involved in the update of the minimum and maximum color values. Parameter $\alpha \in (0, 1)$ controls the update of the texture information. Threshold T_{bw} is used to verify if the updated mode can be considered a reliable background mode. An hysteris scheme is used to update the model weights according to three different parameters: the weight decreasing factor α_w^d which is proportional to the constant factor α_w, on the constant τ and on the maximal weight $\hat{w}_k$. The larger the values of τ and $\hat{w}_k$, the smaller the value of α_w^d and the slower the weight decreases. Modes observed for a sufficiently amount of time see their chances of being forgotten reduce (background models occluded by stationary objects). Details about these parameters and how color and texture distances are computed during mode comparison against color and texture measures at **x** are found in [152].

1.1.5.2 Optical flow

Optical flow is the study of the motion field that exists between two image frames taken at two different times (t and $t + \Delta t$). Such motion field is caused by the relative changes in position and perspective between an observer (any type of sensor such as an eye or a camera) and the scene. In 1981 Horn and Schunck proposed a global method which densely estimates the field of motion between two images [66]. In the same year, Lucas *et al*. proposed a method of differences to

estimate the motion in a local feature neighborhood (popularly known as *KLT*) [90]. *KLT* is more robust to noise and other types of difficulties than Horn and Schunck's method. Because *KLT* is sparse it is faster than Horn and Schunck's which is global. As such, *KLT* is better adapted to address the problem of robust object tracking in real-time that we are dealing in this thesis.

KLT decomposes the problem in two different ones:

1. determining the translation of the window image which surrounds a particular feature point [133] and
2. modeling the linear image deformation [124, 119],

the solution to both problems is based on a matching criterion such as the sum-of-squared-difference (*SSD*) and solving a Newton-Raphson-like optimization problem. Initially, feature windows are detected by feature points detection algorithms (as explained in 1.1.4) based on their texturedness or cornerness. These attributes are measured by their standard deviation in the spatial intensity profile, the presence of zero crossings of the Laplacian of the image intensity and the circular pattern around the point as for the *FAST* corner detector in [119].

A sequence of 2D images is represented in general by two spatial coordinates x and y and a temporal coordinate t. The image signal is then denoted as $I(x, y, t)$. Images taken at near instants are usually strongly related to each other, this correlation is usually expressed as patterns that move in an image stream. The goal of *KLT* algorithm is to identify how these patterns move. Considering this, the image function $I(x, y, t)$ is not completely arbitrary, up to some point it satisfies the following equality

$$I(x, y, t + \Delta t) = I(x - \xi, y - \eta, t). \tag{1.32}$$

The amount of motion $\mathbf{d} = (\xi, \eta)$ represents the displacement vector of the point $\mathbf{x} = (x, y)$ between time instants t and $t + \Delta t$. In most of the cases, different regions in an image are completely unrelated, thus the motion vector $\mathbf{d}$ is a function of x, y, t and τ. In figure 1-13 the displacement $\mathbf{d}$ is depicted as the template (Mario) moves from t to $t + \Delta t$.

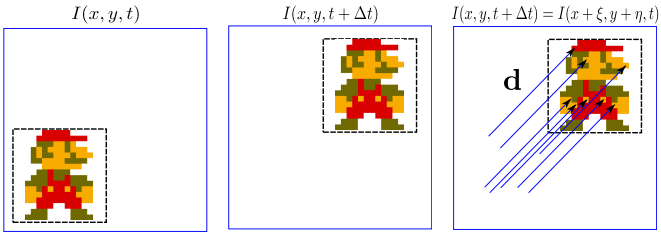


Figure 1-13: Translation optical flow. Point correspondence is found easily for corner points but difficult for edges or homogeneous regions.

Unfortunately, the constancy property expressed in (1.32) is violated in many situations. Images from real world scenes are not frequently planar, most of the time, they are 3D objects located at different depths, their boundaries frequently occlude each other and if not enough, photometric phenomena on their surfaces can drastically affect the appearance captured by the camera depending on the viewpoint. Those problems are illustrated in **Figure 1-14**.

For a single pixel to be tracked it is required that it has a very distinctive brightness with respect to all of its neighbors. This precise value can change due to noise or illumination changes. It is often hard or impossible to determine where a pixel went from one frame to another based only on one local information. *KLT* method does not track single pixels but

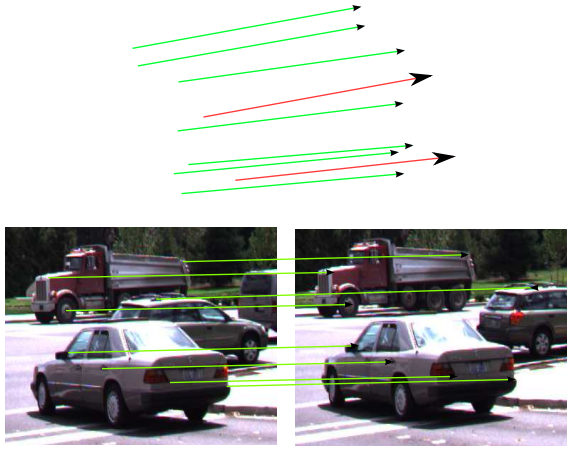


Figure 1-14: 2D images sequences from real world 3D objects generally present occlusions and reflectivity phenomena such as specular reflections violating equation (1.32) assumptions.

windows of pixels. Not every window can be tracked reliably, only windows containing enough texture can be considered. In addition, different points within a window may behave differently, the window may be along an occluding boundary, so that points belonging to different objects move at different velocities. In the worst case, they may even disappear or reintroduce to the image after suffering occlusion. Monitoring the evolution of the residue (sum of squared errors, *SSE*) allows the detection of possible failures.

Optical flow is useful for many applications such as shape from motion and object tracking. The flock of trackers or median flow algorithm (*FoT* or *MF*) described in **Section 2-7** details how to use the results from the KLT optical flow to robustly estimate the displacement of a target. **Appendix A** contains more details about how the optical is computed for simple planar translations and other more complex geometric transformations such as affine maps.

1.1.6 Range information

Range images or *depth maps* can be formed by multiple cameras *stereo-vision* or by *active* methods that project some sort of light forming an special pattern.

In *stereo-vision* a 3D point p being viewed from two cameras whose relative positions is encoded by a rotation and a translation. Fusing the pictures recorded by both cameras and exploiting their *disparity* allows *stereo-vision* algorithms to estimate depth. *Stereo-vision* requires two processes, the first is the *fusion* of the features observed by the set of cameras and the *reconstruction* of their three-dimensional image. However, fusing the set of images turns out to be a very expensive process in computational terms since each picture tipically consists of millions of pixels, with thousands of feature points and edges to match.

Active methods avoid these costs using rays of light and their intersection to estimate the depth of each pixel in the image. Methods based on the triangulation of laser reflections on surfaces exists since the early 1970's. A laser and a pair of rotating mirrors are used to perform a surface sequential scan. Other methods, based on the same principle gain accelerate the scanning of the scene forming a plane of light using cylindrical lenses. The main drawbacks of active triangulation are that the technology is relatively slow and that at some points the laser spot may be hidden from the camera causing undefined regions in the resulting image, some inaccuracies are expected caused by specular reflections, because the reflected light

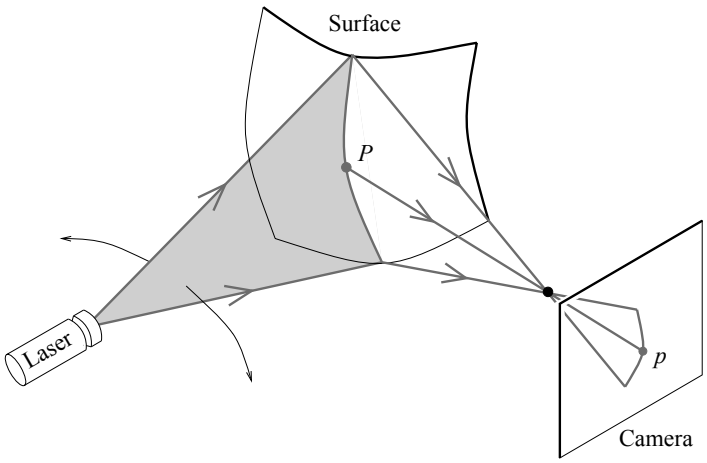


Figure 1-15: A range sensor scanning the surface of an object projecting a laser plane and calculating the triangulations with the aid of a camera (image from [45]).

will not arrive the camera unless it happens to lie in the direction of the mirror direction. Even worse, mirror reflections can find secondary paths which arrive to the camera creating false depth measurements.

Depth-maps are advantageous because pertinent information related to the geometry of the objects and the scene is already provided. Classical image processing methods based on brightness and color information need to be mathematically adapted to deal with range images. Here, contiguous patches with similar shape are grouped using special segmentation methods based on differential geometry used to identify surface discontinuities (*e.g.* step and roof edges). To some extent depth-map features are more robust and invariant to viewpoint changes than images from photographs. They are extremely useful to determine occlusion boundaries and silhouettes and to separate objects from background. Additionally, because range images doesn't have to deal with texture, color or illumination variations it is easier to do realistic simulations in range images than with ordinary photographs, opening the possibility to train accurate classifiers without over-fitting directly from simulations.

Two-dimensional light patterns are now capable of acquiring depth images in real-time and inexpensively. An example of this is the Kinect, a popular video-game device used by Microsoft Xbox 360 and developed by Primesense[2] which provides 30 depth-images per second at a very reasonable price (less than 90 USD in 2013). Depth-images will continue to gain importance in the computer vision and digital image processing communities, as this technology will be soon integrated in other popular consumer electronics products such as digital cameras and smart-phones.

This section we provided a panoramic view of the different local interpretations that a pixel and its neighbors can have in different contexts. Most of the presented local feature detectors and descriptors are based on masks and operators that produce a stronger signal when the presence of a particular chosen type of feature is found. Feature detector operators progressively scan the whole image. The resulting responses are then encoded according to their strength to represent a pixel in the response image. These operators are generally small-sized prototype masks representing the features that we are looking for, the output is a measure of correlation and the goodness of the fit of the prototype to the local image neighborhood. In the next section we will focus on the problem of representing objects using some of the low-level features discussed here.

[2] http://www.primesense.com/

1.2 Object representations

Objects representations are based on cues computed at a pixel level, and combine them by structuring them or cumulating them, in order to provide a relevant signature of the object, which would be preferably unique and therefore separable from other object classes. Robust representations need to consider multiple cues with a high degree of redundancy, carrying shape and appearance information at the same time. The best object representation is context dependent: objects which appear very small in the image don't require a detailed (and complicated) appearance models (*i.e.* a blob's location and size are often enough) while full detailed objects benefit from appearance representations. Single points (centroids) are only suitable for very small-sized objects, bigger ones need to be better represented using their most interesting points (key-points) or a collection of primitive geometric shapes such as a rectangles, ellipses and active contours that register their position, orientation, scale and non-rigid deformations.

Simple motion models such as translations, affine maps, or projective transformations (*i.e.* homographies) are used to adapt these primitive forms to handle deformations. In theory, rectangles and ellipses are more suitable for rigid objects, while non-rigid objects are better modeled by deformable complex shapes such as contours, skeletal models and articulated shapes. But in many practical cases, due their simplicity and execution time limitations, rectangles and ellipses are employed too.

In the rest of this section some of the most important and frequently used object appearance representations are discussed. One of the simplest alternatives is to estimate the object appearance probability densities in a parametric way such as a Gaussian model or a mixture of Gaussians, or with nonparametric methods such as histograms and Parzen windows. Probability densities are estimated considering the pixels inside the shape model (rectangle, ellipse or contour), and assuming a low correlation these representations perform a separation of appearance features and their spatial distribution. It is clear that this supposition is not true, but this assumption is useful considering the possibility of invariant representations. This invariance comes at the loss of some discriminant power: two completely different objects can be confounded when their probability densities are similar enough. Templates (sub-images) carry both spatial and appearance information, but they only encode the appearance from a single view, limiting the degree of allowed variations on the object's pose. Multiple view appearance models blend together multiple different views of an object using principal component analysis (*PCA*) and independent component analysis (*ICA*) used for shape and appearance information. Active contours such as snakes take a local feature map an treat its response as a landscape on which a deformable parametric curve $r(s), 0 \leq s \leq 1$ slithers. Finally, our revision of possible object appearance representations finishes with the covariance-based models that measure feature maps inter-correlations such as colors, brightness, gradients orientation, optical flow and spatial coordinates within the object area, multiple views can be averaged with this approach too, but it is necessary to respect some geometrical properties (see **Section 1.2.5**).

1.2.1 Sub-image representations

The simplest way to represent an object is to consider its image, that is the matrix of pixels which represents the object (see **Figure 1-16-(a)**). Here a direct representation of the appearance pattern is obtained by writing down the intensity or color at each pixel, in some defined order relative to fixed point in the image (commonly the top-left corner).

Other features such as image gradients, Gabor responses, LBP codes and color invariants can be considered too. Feature selection is extremely important since simple image intensities are very vulnerable and sensitive to illumination changes. Unfortunately, there is no image feature suitable for all the variety of problems and for all imaging conditions and this selection as many others in computer vision is context-dependent (see **Figure 1-16-(a)**).

Sub-image representations are based on very basic techniques borrowed from digital signal processing and image processing communities. While these representations are simple and straightforward, their application leads to numerous inconveniences *e.g.*, the total number of calculation grows in proportion to the number of pixel inside the object's shape, it is not robust to object occlusions and depth discontinuities, changes in scale and orientation are not considered. One typical inconvenient of representing a full object depending on a single region is that the possibility of finding a depth discontinuity is higher for bigger tracking regions.

1.2.2 Principal Component Analysis (*PCA*) and Linear Discriminant Analysis (*LDA*)

Turk and Pentland reduced the dimension of the appearance space using a related vector space model for images [138]. With their *Eigenface* approach Principal Component Analysis (*PCA*) is used to estimate the principal directions of variation in the space of all face images (or category space being considered), reducing the high-dimensional pixel lists to a much more compact encoding which highlights the key appearance attributes. *PCA* turns a set of possibly correlated variables into a smaller set of uncorrelated variables, this way, only few dimensions hold most of the information. When *PCA* analysis is performed, images are decomposed into a linear combination of vectors accounting for the greatest variance in data (principal components), only the eigenvectors with the greatest eigenvalues are kept and the object is represented by the set of coefficients of this linear combination (see **Figure 1-16-(b)**).

Some years later, Belhumeur *et al*. proposed to optimize class separability by working instead in a subspace obtained by Fisher's Linear Discriminant Analysis (*LDA*). Because *LDA* explicitly attempts to model the difference between the classes of data, the resulting method known as *Fisherfaces* [11] improves the discrimination capabilities of *PCA*-based methods and is able to deal with specific discrimination tasks such as distinguishing people with mustaches, or people wearing glasses from the rest.

As with sub-image (template) representations, vector space models require closely aligned 2D patterns. Template and sub-image alignment is demanded to ensure a fair comparison of the object attributes *e.g.* noses and eyes should be positioned in the right place when comparing images of frontal faces.

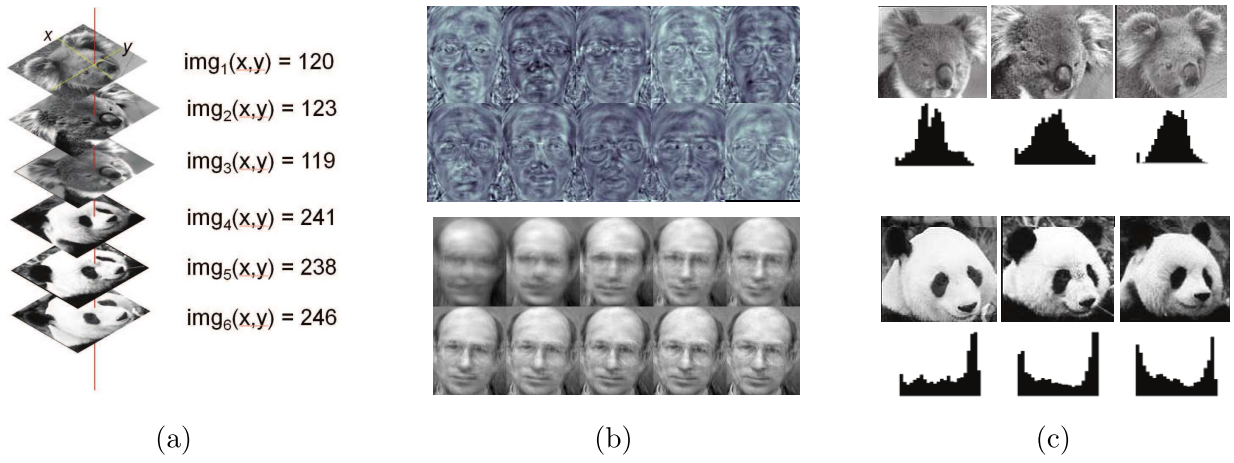


Figure 1-16: (a) In the sub-image (template) representation, an image of size ($N = W \times H$) is represented by a list of N gray-values or colors. Each sub-image is a point in the N-dimensional space. (b) Using *PCA* or *LDA* analysis, the original N-dimensional space is reduced to a vector space where images are represented by linear combinations of the vectors of greatest variability. (c) Using histograms, an holistic description is formed by measuring the distribution of pixel intensities, colors, gradients, etc. Images taken from [130].

1.2.3 Histogram representations

One alternative to the aforementioned representations is to form a simple and holistic description of the set of pixels by measuring the distribution of their intensities, colors or gradients for instance. An histogram is a discrete function that counts the number of observations (values) that lie into a set of disjoint categories (known as bins), providing a description of the data distribution which is very convenient for evaluating image attributes. Intensity histograms are used for basic image processing tasks such as image equalization where image contrast is improved avoiding the concentration of pixels in a small range of values. Histograms are also used with other type of images to estimate dominant colors, textures and gradient orientations. They represent an important tool in computer vision because they are not affected by changes of shape, image rotations and scale (see **Figure 1-16-(c)**).

For images containing colors, gray levels or angles it is unpractical to map each possible value to an array (the resulting array would be too large and sparse). A more convenient option is an array of B histogram bins (*i.e.* buckets), where each bin stores the number of pixels with values inside a predefined interval of size h.

Histograms are formally defined as

$$\hat{f}(x) = \frac{1}{nh}(\text{\# of pixels in the same bin as } x) \tag{1.33}$$

where n is the total number of pixels in the image, X_i for $i = 0 \cdots n-1$ their values and h represents the bin width.

Typical categories of histograms commonly used in computer vision are:

a) **Intensity histograms**: The histogram of a monochrome image (*i.e.*, intensity image) provides a representation of the frequency of occurrence of each gray level value in the image. Concentration of pixels in the array's lower end is an indicator of a predominantly dark image while bright images concentrate their pixels in the opposite end of the array. Similarly, low contrast images are identified by a high concentration of pixels in a narrow range of levels, on

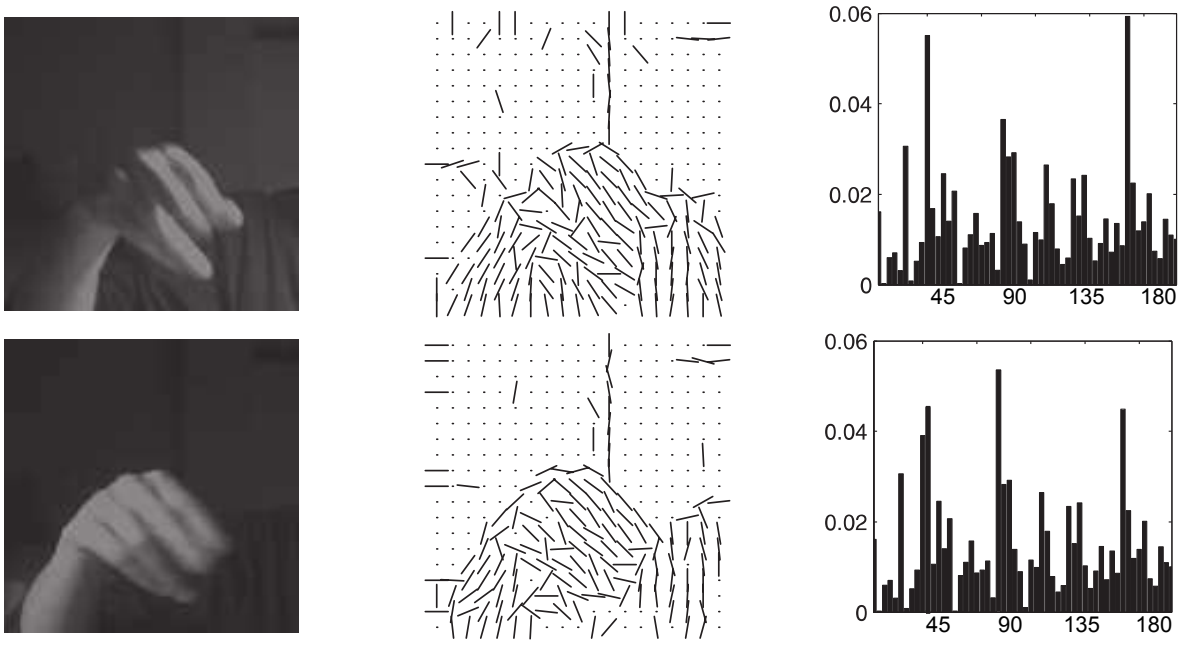


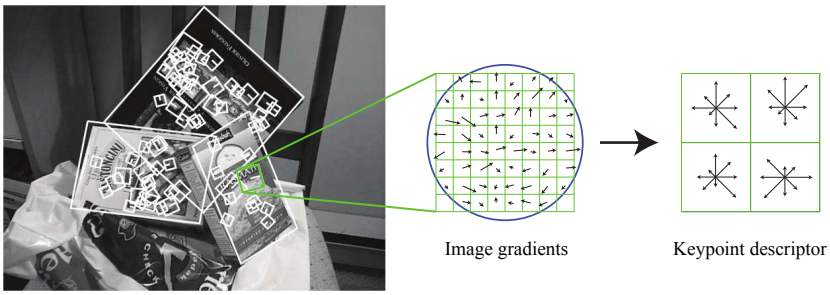
Figure 1-17: Pixels are severely affected by changes in illumination, gradient orientations are less sensitive and more robust to those changes.

the contrary, high contrasted images exhibit a bimodal histogram with a clear separation between the two predominant modes. These histograms are not invariant against contrast changes but provide a good robustness against geometric variations.

b) **Gradient histograms**: Pixel intensities are very sensitive to changes in scene lighting. Working with hand gesture recognition, Bichsel observed [14] that local orientation measurements are less sensitive to illumination changes, he proposed to estimate the local orientation of the dominant edge orientation to build a descriptor of gestures. The idea of representing local object appearance and shape by the distribution of local intensity gradients or edge orientations inspired more recent and popular methods such as the *SIFT* key point descriptor [89] (introduced in 1.1.4) and the histograms of oriented gradients (*HOG*) developed originally for pedestrian detection [34] (see **Figure 1-18**). Note that the precise information of the corresponding gradients and/or the edge positions is discarded when calculating gradient distributions. In practice, these methods sub-divide the image window into small regions (*cells*) where each cell stores a local 1D histogram of the gradient directions or edge orientations corresponding to the pixels inside the cell.

Gradient histograms are of a tremendous importance in computer vision, the popular histogram of oriented gradients (*HOG*) descriptor concatenates a series of overlapping gradient histograms to construct a vector descriptor, this algorithm presented by Dalal and Triggs [34] is currently one of the most important algorithms for object recognition and other related tasks. To compute the *HOG* descriptor a sub-window is first divided into small spatial *cells* and each one produces an histogram based on the gradient values and their orientations for all pixels within the cell. The orientations in the 8×8 sized cells are grouped into 9 orientation bins. *HOG* descriptors are already robust to small shifts and rotations, to provide robustness to illumination effects, local gradient responses are contrast-normalized using a block-based measure. This normalization is done before entering the information into the histograms.

(a)



(b)

Figure 1-18: Popular methods use gradient-based histograms as descriptors, *e.g.*, (a)*SIFT* local keypoint descriptor and (b) R-*HOG* descriptor.

HOG descriptors do a weighted counting of the orientations in a cell. Gradient magnitudes at each location are compared to other locations in the same cell to discover their relative relevance. Contrasting with *SIFT* descriptors, here normalization is done with respect to the nearby gradients only, and due to the overlap between regions, each single gradient location contributes to several different histograms reducing to the extreme the likelihood of missing low-contrast boundaries. If $\|\nabla I_{\mathbf{x}}\|$ represents the gradient magnitude at the location $\mathbf{x}$, and $\mathcal{C}$ represents the cell whose histogram we are interested to compute, then $w_{\mathbf{x},\mathcal{C}}$ is the weight assigned to the orientations at $\mathbf{x}$ for this cell. Weights are chosen as

$$w_{\mathbf{x},\mathcal{C}} = \frac{\|\nabla I_{\mathbf{x}}\|}{\sum_{\mathbf{u}\in\mathcal{C}} \|\nabla I_{\mathbf{u}}\|} \tag{1.34}$$

which favors gradients that are large when compared to its neighbors. The effect of this operation is an enhancement of the objects outline curves.

The final *HOG* sub-window descriptor is formed by the normalized histograms from the dense grid of overlapping blocks.

c) **Color histograms**: Under some circumstances color-based descriptions provide a higher discriminative power helping to recognize objects by their color characteristics rather than merely depending only on brightness and gradients. However, illumination and recording conditions may affect the observed colors in a complicated way and photometric invariance methods are often required [54]. Color description methods are commonly based in different multidimensional data classification techniques such as color image segmentation *i.e.*, extracting significant colored regions in an image or object. In the trichromatic representation, each pixel is associated to a point in the 3D color-space. Pixels which belong to the same region usually have similar properties and form compact point-clouds in the color space. In histogram-based methods, spatial pixel distribution doesn't matter and color distributions are considered only. Some operators process colored pixels to obtain photometrically invariant information, the problem is that under some circumstances such as low saturation and illumination, these invariants are very unstable.

Unidimensional histograms are based on the supposition that homogeneous color regions can be easily identified by mode manifestations inside each 1D component histogram. By projecting color information into a sub-space of inferior dimension it is possible to reduce the computational consumption. The problem is the risk of underestimating information loss that occurs when the algorithm ignores the different modes that appear in the original representation. Multiple techniques exist in the literature to manipulate 1D histograms, their main differences reside in the selection of the appropriate color components, the extraction of modes from the histograms and the determination of the most representative component. A severe inconvenient of 1D color histograms is the difficulty to perform color class separation: two separable classes in the 3D space may become impossible to separate when projected to the 1D space. As a result, different regions are involuntary grouped into a single region.

Color histograms require great amounts of memory, typically RGB colors re encoded using 256^3 values. A simple and naive solution consist in reducing the size of the histogram by considering only the most significant bits of each component. This is equivalent to reducing the number of quantization levels and performing a type of pre-segmentation. The problem with this method is that it does not consider the original distribution of the color pixel values, other techniques prefer to first analyze the 3D histogram to identify high density cells from the lower density ones. Next, a set of connected high density cells is grouped into a class. This technique requires a threshold which has a significant impact on the performance of the method. Indeed, two different classes can be accidentally grouped into a same class if it is too low, and many classes can be ignored if it is too high.

Two important choices have to be made when constructing and histogram. The first one is the bin-width h and secondly, the precise location of the bin edges. Both decisions can greatly affect the resulting histogram. Using kernels it is possible to avoid histogram sensibility to bin edges positioning. Then equation 1.33 becomes

$$\hat{f}(c) = \frac{1}{n} \sum_{i=1}^{n} \sigma_{C_i}^{-1} K \left(\frac{c - C_i}{\sigma_{C_i}} \right), \tag{1.35}$$

where c represents one of the color channels contained in x, and $\{C_0, \cdots, C_n\}$ is the set of i-bins on that channel, σ_{C_i} is the standard deviation of each bin. Unstable pixel values in channel c are associated to very smooth (widespread) kernels, while well-defined and stable pixel values are associated to a very narrow one (see **Figure 1-19**).

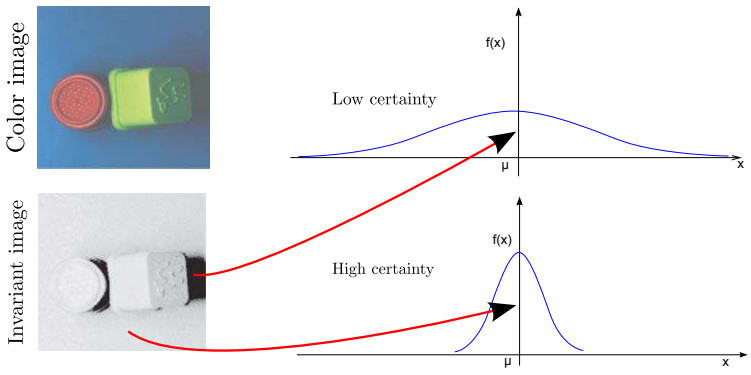


Figure 1-19: Robust histograms are constructed using color uncertainty to derive the parametrization of a variable kernel. Kernel sizes are steered by the amount of uncertainty of the color invariant values.

1.2.4 Segmentation and active contour representations

In a natural environment, it is very unlikely to find objects of very simple geometric shapes such as circles, rectangles and ellipses. Irregularly curved complex shaped objects are more common in the real world, to avoid model statistics contamination it is often desirable to find the boundary that separates them from the background. This boundary can be found by segmentation-based methods that assign a label depending on pixel properties such as brightness, texture, color, location and motion. Segmentation is one of the most widely studied problems in computer vision, the most important type of segmentation algorithms are k-means [91], region splitting and merging [27], mean-shift [28], active contours (*e.g.*, snakes) Kass *et al*. and level sets [103], normalized-cuts and binary Markov random fields together with graph cuts [18].

Region splitting (*divisive clustering*) is the process of finding finer image regions depending on the concentration of the information. The algorithm starts with a single region, it estimates its histogram and determines a threshold that best separates the larger peaks on it. The process is repeated for each resulting region until all the histograms are uniform enough or below a certain size. Region merging (*agglomerative clustering*) does the opposite than region splitting, these algorithms start with pixel-sized regions and link them together based on their similarity (*e.g.*, average color difference) and their region size. Region merging methods can be used as a pre-processing step that obtains pixel groupings (or *super-pixels*) [98] to robustify and accelerate the execution of higher-level algorithms such as optic-flow, stereo matching and object recognition.

One of the first methods to estimate the parametric curve that surrounds a target is the method of *snakes* [76] developed in 1988 by Kass *et al*.. This method minimizes an energy function by adjusting a two-dimensional spline curve that evolves towards strong operator responses, for contrast enhancement filters, snakes are attracted towards the edges clinging to high energy filter responses.

The parametric curve is defined by $r(s),\ 0 \leq s \leq 1$ and it is calculated by maximizing the *external* potential energy function $F(r(s))$, where $F(\mathbf{x})$ is the operator response at pixel $\mathbf{x}$. This parametrization is counterbalanced by the *internal* potential energy function that privileges and preserves the smoothness of the curve

$$\underbrace{\left(\frac{\partial(w_1 r)}{\partial s} - \frac{\partial^2(w_2 r)}{\partial s^2}\right)}_{\text{internal forces}} + \underbrace{\nabla F}_{\text{external force}} = 0 \tag{1.36}$$

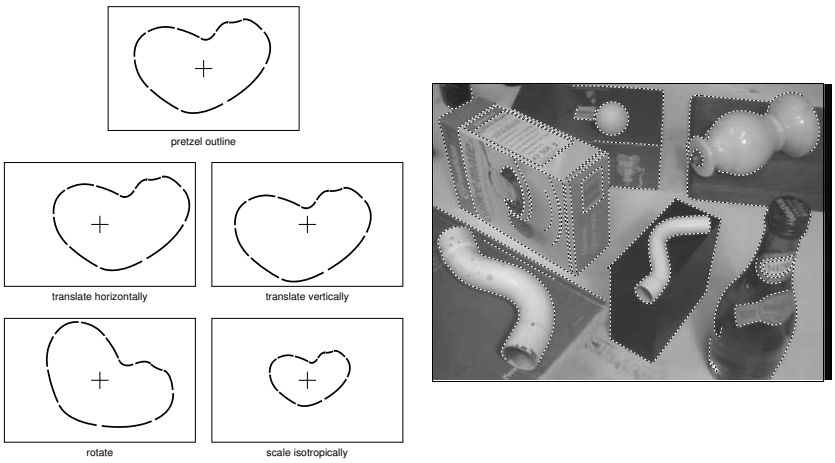


Figure 1-20: The geometrical transformations experience by the whole object can be modeled and simplified by determining the transformation of the active contours parametrization model instead (see examples on the left). Parametric spline curves are attracted toward edges operator responses (image on the right). Image from [130].

coefficients w_1 and w_2 are always positive and control the elasticity and stiffness of the snake.

Patterns of snake models are matched to an image by searching for the value of the parameter vector that minimizes the external energy. The internal energy is used as a regularizer which favors certain shapes.

Active contours can be applied for modeling dynamic images too (temporal sequences) by defining an additional layer of likely object motions and deformations. In this situation, the active contour model is expressed as $r(s,t)$ and the feature maps evolve with time $F(t)$. A method to estimate the temporal evolution of the contour was later proposed by Terzopoulos and Szeliski in 1992 based on the Kalman filter to model the uncertainty of the changes using a Gaussian model. After this, Isard and Blake proposed in 1998 the method of *condensation* [68] that introduced *particle filters* to the computer vision community for modeling multi-modal distributions. Using *level sets* it is possible to avoid the estimation of the parametric curve, instead, the *zero-crossings* of a *characteristic* function are used to define it. Level sets evolve to fit and track objects of interest by modifying the *embedding function*. Some level-set methods are based on image gradients and others re-cast the problem into a segmentation framework, where an energy function measures the consistency of the image statistics (*e.g.*, color, texture, motion) inside and outside the segmented regions.

1.2.5 Matrix based descriptors

For many years, the raw pixel values of several image statistics such as color, gradient and filter responses were used because they are the simplest choice for image features. The Lucas-Kanade (*KLT*) optical flow algorithm (**Section 1.1.5.2**) developed in the 1980's and 1990's belongs to this type of methods. Raw pixel matching is problematic because pixel values are not robust in the presence of illumination changes and nonrigid motion. Furthermore, because of the high dimensional representation (each pixel acts as a dimension), matching time grows with the size of the region and the number of features considered. Histogram methods (**Section 1.2.3**) provide a natural solution to these problems because a region is represented as a nonparametric estimation of the joint distribution. Histograms are widely used for nonrigid object tracking, texture representation, matching and other problems in the field of computer vision. Their great disadvantage is that histogram size grows exponentially with the number of features considered.

As alternatives to the traditional methods just mentioned, computer vision and computer graphics are increasingly relying on machine learning and information-theoric methods. On the other side, information-theoric methods include the still developing field of *Computational Information Geometry* as a novel paradigm to perform high-fidelity data analysis using the language and thinking of geometry.

Among the many the existing types of data representations, symmetric positive definite matrices (*SPD*'s) are of particular interest to this field. *SPD* matrices are gaining importance because it has been observed that in very different applications and with very heterogeneous sources such as Diffusion tensor imaging (*DTI*) [143], brain-computer interfaces (*BCI*) [9] and conventional imaging the information can be modeled and with great convenience and with a very similar fashion with covariance matrices which are just a particular case of *SPD* matrices. In computer vision, Tuzel et al. [139] proposed to use the covariance matrices obtained from several image statistics computed inside a region of interest as the region descriptor (known in the literature as covariance descriptor). In comparison to histogram representations, covariance descriptors dimensionality is much smaller. Even better, covariances can be calculated very fast using integral images and the cost is almost independent of the size of the region. Covariance matrices (*SPD*'s in general) are not elements of the Euclidean space, they are points in a Riemannian manifold. As such, classical machine learning algorithms for clustering and classification need to be adapted to work in Riemannian geometry.

Covariance descriptors require basically two types of operations: 1) measuring distances between two points in the manifold and 2) barycenter calculation to have a model that represents all the samples coming from the same class. Knowledge on Riemannian geometry is required to understand these concepts. In the rest of the section, it is explained how to compute the covariance descriptor given an image and a region of interest, and then different methods for measuring distances and updating the models are introduced. This section describes the principles of the method, it explains an efficient computation of the descriptor together with some examples of feature combinations frequently used for this type of descriptors.

Covariance matrices as image region descriptors

Let I represent a luminance (grayscale) or a three dimensional color image (other types of images can also be considered: infrared, depth images, etc.), and let F be the $W \times H \times d$ dimensional feature image extracted from I

$$F(x, y) = \phi(I, x, y) \tag{1.37}$$

where ϕ is any d-dimensional mapping forming a feature combination for each pixel including features such as intensity, color (in any color space), gradients, filter responses, or any possible set of images obtained from I (feature configurations are covered in **Section 1.2.5**). Let now $\{\mathbf{z}_k\}_{k=1\cdots n}$ be a set of d-dimensional points inside the rectangular region $R \subset F$ (region size is n). The region R is represented with the $d \times d$ coavariance matrix

$$\mathbf{C}_R = \frac{1}{n-1}\sum_{k=1}^{n}(\mathbf{z}_k - \boldsymbol{\mu})(\mathbf{z}_k - \boldsymbol{\mu})^T \tag{1.38}$$

where $\boldsymbol{\mu}$ is the mean vector of the points.

The covariance matrix is a $d \times d$ square matrix which fuses multiple features naturally measuring their correlations. Elements in the main diagonal represent the variance of each feature, while elements outside this diagonal represent the

correlations. Thanks to the averaging in the covariance computation, noisy pixels are largely filtered out which contrasts with raw-pixel methods. Covariance matrices are low-dimensional compared to other descriptors, and due to symmetry $\mathbf{C}_R$ has only $(d^2 + d)/2$ different values. The covariance descriptor $\mathbf{C}_R$ does not use any information regarding the ordering or the number of the points implying a certain scale and rotation invariance. Nevertheless the covariance descriptor ceases to be rotationally invariant when orientation information is introduced in the feature vector such as the norm of gradients with respect to x and y directions.

Matching and tracking applications are forced to calculate multiple covariance descriptors in overlapping regions where a high redundancy of operations is expected. Viola and Jones and Tuzel *et al*. use a very clever trick that allows to compute overlapping region averages supper fast. This method is popularly known as the *integral images* or the *summed area tables* method. Integral images are popluar as they are used in many computer vision methods in different contexts (*e.g.*, Viola and Jones face detector [144] and Bay *et al*.'s Speeded Up Robust Features **SURF** [10]). When constructing the integral images accumulations are performed first over the whole image or over a region of interest and the integral images which result from this accumulation are re-utilized to calculate the feature averages required by equation (1.38).

Let i be an image of size $W \times H$ and let I be its correspondent integral image of the same size. The value of each pixel in I results from the accumulation of all the pixels inside the rectangle defined by the image's upper left corner and the pixel of interest in i. Mathematically the value of $I(x, y)$ is

$$I(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y'). \tag{1.39}$$

Once equipped with the integral image I, any rectangular sum can be computed in constant time with only four references, the method is graphically described in **Figure 1-21**.

$$\sum_{\substack{A(x) < x' \leq C(x) \\ A(y) < y' \leq C(y)}} i(x', y') = I(C) + I(A) - I(B) - I(D). \tag{1.40}$$

For the sake of simplicity, many details about the covariance descriptor computation are omitted here, a thoughtful discussion is provided in **Section B.1** while some implementation aspects about the integral images method (precision and parallelization) will be discussed in **Section 5.2.2**.

Covariance descriptor feature spaces

The information considered by the covariance descriptor should be adapted to problem at hand. Covariance descriptors have been used in computer vision for object detection [140], re-identification [6, 114] and tracking [109]. The recommended set of features to use depends significantly on the application and the nature of the object: tracking faces is different than tracking pedestrians for example. The set of textures present in faces is completely different than the set of textures found in pedestrians, or vehicles. Color plays an important role when tracking/re-identifying pedestrians or cars because the enormous variability of colors they may have, but color has less significance for faces because they are very limited in the set of colors they usually have.

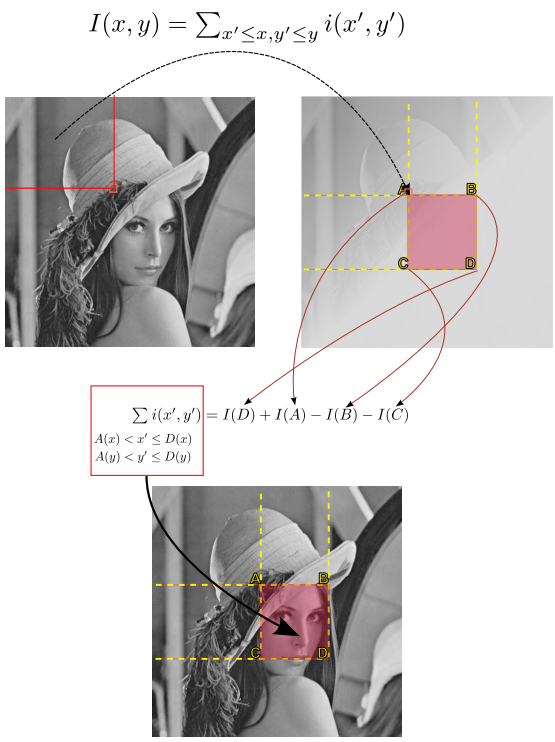


Figure 1-21: Integral images concept.

Table 1.1 displays a summary of the different feature combinations that could be found when using covariance descriptors in computer vision. Covariances obtained from pixel locations (x, y) are the same for all regions of the same size, but still they are important because they give a hint of the other features spatial relationship. Observing that the original features of Tuzel *et al*. in [139] behaved poorly when applied in face recognition problems Pang *et al*. proposed in [104] to use Gabor wavelets to describe faces in a database. These filters have similar responses to the mammalian simple cortical cells, exhibiting strong characteristics of spatial locality, scale and orientation selectivity. Features based on these filters are expected to carry more important information than the first and second order gradients boosting the discriminative capacity of the covariance descriptor. The 2-D Gabor kernel is a product of an elliptical Gaussian and a complex plane wave

$$\varphi_{u,v}(z) = \frac{\|k_{u,v}\|^2}{\sigma^2} e^{(-\|k_{u,v}\|^2 \|z\|^2 / 2\sigma^2)} \left[e^{ik_{u,v}z} - e^{-\sigma^2/2} \right] \tag{1.41}$$

where u and v govern the orientation and scale of the kernels. The wave vector $k_{u,v}$ is defined as

$$k_{u,v} = k_v e^{i\phi_u} \tag{1.42}$$

where $k_v = k_{max}/f_v$ and $\phi_u = \pi u/8$. The maximum frequency k_max is usually set to 5 in the field of face recognition, f_v is the spacing between kernels in the frequency domain. The family of Gabor kernels are constructed by two parameters: orientation $u \in \{0, \cdots, 7\}$ and scale $v \in \{0, \cdots 4\}$. The Gabor features that go into the covariance matrix are

$$g_{uv}(x, y) = |I(x, y) * \varphi_{u,v}(x, y)| \tag{1.43}$$

representing the magnitude of the convolution of $\varphi_{u,v}$ with I.

Table 1.1: Features considered by the covariance descriptor depending on the application.

Application	**Feature set** $\phi(I,x,y)$
Face tracking and recognition [104]	$\left[\begin{array}{ccccccc} x & y & \lvert I_x\rvert & \lvert I_y\rvert & \lvert I_{xx}\rvert & \lvert I_{yy}\rvert \end{array}\right]$
	$\left[\begin{array}{ccccccccc} x & y & I & \lvert I_x\rvert & \lvert I_y\rvert & \lvert I_{xx}\rvert & \lvert I_{yy}\rvert & \theta(x,y) \end{array}\right]$
	$\left[\begin{array}{ccccccc} x & y & I & g_{00}(x,y) & g_{01}(x,y) & \cdots & g_{74}(x,y) \end{array}\right]$
Pedestrian detection [140, 151]	$\left[\begin{array}{ccccccccc} x & y & \lvert I_x\rvert & \lvert I_y\rvert & \sqrt{I_x^2+I_y^2} & \lvert I_{xx}\rvert & \lvert I_{yy}\rvert & arctan\frac{\lvert I_x\rvert}{\lvert I_y\rvert} \end{array}\right]$
	$\left[\begin{array}{ccccccccc} x & y & \lvert I_x\rvert & \lvert I_y\rvert & \sqrt{I_x^2+I_y^2} & arctan\frac{\lvert I_x\rvert}{\lvert I_y\rvert} & \mathbf{G} & \sqrt{\mathbf{G}_x^2+\mathbf{G}_y^2} \end{array}\right]$
Pedestrian tracking [140, 109, 6] and [114]	$\left[\begin{array}{ccccccc} x & y & R & G & B & \lvert I_x\rvert & \lvert I_y\rvert \end{array}\right]$
	$\left[\begin{array}{ccccccccc} x & y & R & G & B & \lvert I_x\rvert & \lvert I_y\rvert & \lvert I_{xx}\rvert & \lvert I_{yy}\rvert \end{array}\right]$
	$\left[\begin{array}{ccccccc} x & y & H & S & V & \lvert I_x\rvert & \lvert I_y\rvert \end{array}\right]$
	$\left[\begin{array}{cccccc} x & y & R & G & B & \mathrm{Var}_{LBP} \end{array}\right]$

1.3 Conclusions

This chapter has presented the basics of the image formation process and different cues useful for higher level computer vision algorithms such as: luminance, color, spatial gradients, temporal gradients (movements) and range images. The image operators presented in this chapter are just a sample of the immense amount of algorithms developed by the computer vision community to extract and analyze all this information.

Most of the operators and algorithms described on the first section of the chapter are considered low-level because they are applied uniformly and without distinctions to all the pixels in an image searching for interesting textured locations, highlighting zones of high luminance or color contrast, or modeling how pixel values evolve in time denoting the presence of movement.

In the second section some medium-level algorithms and operators used to build object representations were discussed. While basic sub-image representations that compare patches pixel-by-pixel can be enough for some cases, these representations are often unable to handle image noise and structure variations caused by scale changes or articulate deformations in flexible objects. Some statistical methods such as *PCA* and *LDA* were presented after, these representations are useful to capture the directions of variation using the eigenvalue decomposition or optimizing the separability of the data by modeling the differences. Histograms representations build an holistic description using the distribution of the pixel intensities, colors, gradients and other features. Histograms are convenient for describing objects because they are invariant to changes of shape, size and orientation. To embed some degree of structure to the histogram representations the popular *HOG* descriptor uses a grid of overlapping histograms. Contour representations are useful because simple geometric shapes such as circles, rectangles and ellipses are uncommon in a natural environment and each type of object usually haves a very characteristic shape, the problem with this representations is the elevated complexity of the operations they often require, as one of the objectives of this thesis is to propose an algorithm that is able to run in real-time in present day embedded

architectures this representations were not considered.

Finally, the covariance matrix descriptor presented in **Section 1.2.5** similarly to the histogram representations it is able to handle nonrigid objects, fusing multiple features without the exponential growth on the descriptor size observed in histogram representations among other advantages makes the covariance matrix descriptor a strong candidate for the real-time multiple object tracking algorithms proposed in this thesis.

Among the pixel cues defined in this chapter, many of them are useful for the continuation of this thesis. Particularly object representations used for detection or matching based on:

- color distributions,
- a collection of feature points with their optical flow,
- covariance matrices of space, color and gradient/texture features,
- histograms of oriented gradientes (HOG) to perform pedestrian detection for mobile or static cameras.

Next chapter is devoted to the following problems:

- **Object detection**: *machine learning* techniques are used to analyze the information contained within a set of *labeled* samples, build generalizations and applying them to classify a set of unlabeled samples.
- **Matching and tracking**: Matching is used to compare the appearance descriptors or on-line representations of each one of the detected targets in a scene. Tracking algorithms propose some high-level techniques to trace the evolution of a target in a scene and strategies defined to handle problems such as occlusions and target crossings.

CHAPTER 2

Detection and matching

Contents

Introduction

In tracking applications, the objective is to estimate the state (*e.g.*, location, size, orientation) of a moving target over time. This process is often subdivided into two other subproblems: detection and matching. Detection deals with the difficulties of generic object recognition *i.e.*, finding instances from a particular object class or semantic category (*e.g.*, humans, faces, vehicles) registered in digital images and videos. Matching methods in turn recognize the identities from objects previously seen in older frames. Generic object recognition requires models that capture the variability of the instances appearances and shapes. Matching algorithms on the other hand analyze particular information and construct discriminative models that help to disambiguate all different instances of the same category to avoid confusions.

Using appearance matching and dynamical constraints it is possible to design tracking algorithms that trace the target trajectories and follow their changes of appearance, location, orientation, scale and shape. A variety of tracking methods have been proposed since the beginnings of computer vision, some of them estimate the target paths following their state evolution using a Bayesian framework (*e.g.*, particle filters, hidden Markov models, conditional random fields), others measure the perceived optical flow to determine object displacements and scale changes (*e.g.*, median flow). Exhaustive appearance-based methods compare a dense set of overlapping candidate locations to detect the one that best fits a model (based on texture, spatial frequency information or any other hint). The number of comparisons is reduced giving some preference to the most likely locations based on the previously observed target dynamics. Local searches based on gradient-descent algorithms can be used too to smoothly detect target displacements but these methods tend to fail if the tracker is

attracted by local minimum.

This chapter opens discussing some of the most popular classifier techniques taken from the machine learning literature to the computer vision field. For instance, **Support Vector Machines** (SVM's) [32] and **AdaBoost** [47] are widely used to tackle the problem of object detection (this is discussed in **Section 2.1**) and activity recognition. Both methods are vectors classifiers that operate on Euclidean spaces, but many mathematical entities in computer vision do not form vector spaces, in turn, they often reside on non-linear manifolds. Symmetric positive definite matrices (SPD) such as the covariance matrix descriptors presented in **Section 1.2.5** are just a good example. Some considerations that are detailed in **Section 2.1.3.1** are thus required to perform covariance descriptor classification. The most common approach consists in computing the tangent space to the manifold at the mean of the data points and obtain an Euclidean approximation of the manifold, but a newer approach presented in [69] represents the internal product between the vector mappings obtained from the SPD matrix space into a pair of vectors located in a high dimensional Hilbert space where Euclidean geometry applies correctly. This technique is used for the particular case of pedestrian detection using covariance matrix classification. Object classifiers are the corner stone of the *tracking-by-detection* algorithms (such as the one proposed in **Section 4.4**), where targets are re-detected every frame (or blocks of frames) and the tracking algorithm is only charged of identifying the targets and tracing their trajectories.

Popular matching operators such as the mean square error (MSE), the sum of absolute differences (SAD) or the Riemannian metric are just some examples of the typical operators, metrics or divergences used to compare a pair of image regions (or their models) and establishing correspondences. These operators are discussed in **Section 2.2** together with other high-level algorithms popular for object tracking applications: the Flock of Trackers (FoT) algorithm (**Section 2.2.2**) and Covariance Tracking algorithm (*CT*) (**Section 2.2.4**) are based on these matching techniques.

2.1 Object detection by descriptor learning

Object detection requires a class model learned from multiple training instances. This model captures the variability of those samples to obtain a generic representation suitable for the detection of unknown instances. Two of the most popular classifier techniques are **Support Vector Machines (SVM)** and **AdaBoost**. They are widely used in object detection and categorization problems. For instance, the Viola and Jones face classifier is based in **AdaBoost** while the Dalal and Triggs HOG classifiers are not just used for pedestrian detection but for the detection of other types of objects too. Both methods operate with vectors inside Euclidean spaces where well-developed learning methods exist. Unfortunately, this is not the case for the problem of classification on Riemannian manifolds where appropriate ways to separate a manifold into two or more classes is still an active subject. An approach to tackle this problem is to map the input patterns on the Riemannian manifold to an Euclidean vector and to apply there any of the well-studied vector classifiers. The performance of this approach can be very low because it fails to identify the inherent structure of the Riemannian manifold. The method of Tuzel *et al*. [140] presented in **Section 2.1.3.1** (page 72) considers the geometrical properties of the Riemannian manifold and obtains better results. This method uses a LogitBoost classifier adapted to the Riemannian manifold thanks to the mapping functions that take a covariance matrix to its local tangent space. A novel technique was presented by Jayasumana *et al*. in [69] where Kernel SVM's are used on the set of SPD matrices Sym_d^+ for training and testing. The authors argue that the iterative approach of [140] which requires a combination of weak learners on different tangent spaces is an extremely

expensive gradient procedure that is repeated at every boosting iteration. Being the cause of a very poor scalability of the algorithm in terms of the number of training samples. Furthermore, the method of Tuzel *et al*. learns classifiers on tangent spaces used as Euclidean approximates of the manifold, while the method of [69] makes use of a rich high dimensional feature space.

The outline of this section is the following: Support Vector Machines (SVM) and AdaBoost are introduced first to briefly describe how pattern recognition and data analysis work as supervised learning models in machine learning. Once these classifier algorithms and their most important details have been introduced we will continue with an explanation of their application for the problem of object detection in computer vision. At the end of the section, the required considerations for covariance matrix descriptor classification using these classifier techniques are introduced.

2.1.1 Support Vector Machines (SVM)

In this subsection a brief introduction to the problem of data classification with Support Vector Machines (SVM's) is given. The very basic concepts of the separating hyperplane, how the separating margin is maximized and kernels methods are discussed. The primal and dual optimization problems are presented too. The aim of this subsection is to present the pattern classification problem and to discuss its geometric properties. How these optimization problems are solved is not discussed here. More comprehensive introductions to SVM's and their implementations can be found in [88] and [17].

One of the fundamental problems of learning theory is data classification. The simplest case of this problem occurs when we are given two classes of objects and then faced with a new object that we are required to assign to the most similar class. The formal definition of this problem is as follows: given the empirical data $(x_1, y_1), \cdots, (x_m, y_m) \in \mathcal{X} \times \{\pm 1\}$, where $\mathcal{X}$ is some non-empty set from which the patterns x_i are taken (often referred to as the *domain*), and where the y_i values are often called *labels*, *targets*, *outputs* or *observations*. Here, only two classes of patterns are presented, thus, this is a *binary classification* problem. More complex problems can occur where the patterns are assigned to many different labels (or classes). The problem of *learning* to *generalize* the information contained by the presented labeled samples (known as *training* samples) and apply it to unseen data points, this means that when we are presented with some new pattern $x \in \mathcal{X}$, the problem consists in determining the corresponding $y \in \{\pm 1\}$ so that in some sense the pair (x, y) is *similar* to the training examples. To achieve this, a notion of similarity in $\mathcal{X}$ and in $\{\pm 1\}$ is required. Comparing binary labels $\{\pm 1\}$ is trivial as they can only be identical or different, but the chosen similarity measure for the input space is a deep question that machine learning algorithms try to solve depending on the context.

A similarity measure of the form $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a function that maps two given patterns and returns a real number that characterizes their similarity. It is assumed that the function k is *symmetric*, this is $k(x, x') = k(x', x)$ for every pair $x, x' \in \mathcal{X}$. This type of functions are called *kernels* in the context of machine learning literature.

A very common and simple similarity measure used in mathematics is the *dot product*, given two vectors, $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$, the dot product operator is defined as

$$\langle \mathbf{x}, \mathbf{x}' \rangle := \sum_{i=1}^{N} [\mathbf{x}]_i [\mathbf{x}']_i, \tag{2.1}$$

which corresponds to the addition of the element-by-element (i-th entries) products. The geometrical interpretation for the dot product operator is that it computes the cosine of the angle formed by the two vectors (if both vectors have been

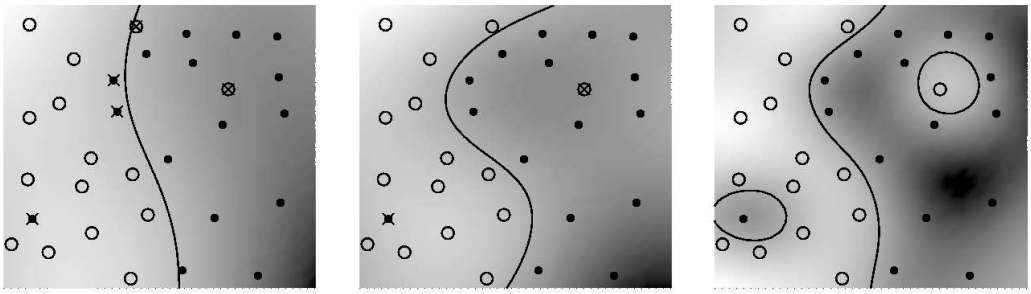


Figure 2-1: A simple example of 2D binary classification. Three models increasing in complexity from left to right are applied. The model on the left miss-classifies a large number of training samples, the more complex one on the right *trusts* the training samples and proposes a solution that is consistent with the whole training set but that may not work well on new points. The model in the middle allows some training samples to be miss-classified looking for a more *general* solution. Image taken from [88].

normalized). The self-dot product of a vector $\langle \mathbf{x}, \mathbf{x} \rangle$ allows the computation of the length (norm) of the vector $\mathbf{x}$ as

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}. \tag{2.2}$$

Similarly, the length of the difference vector represents the distance between the two vectors $\mathbf{x}$ and $\mathbf{x}'$. Therefore, the dot product operators allows us to compute angles, lengths and distances.

Given a vector space $\mathcal{H}$ equipped with the dot product operation and a set of patterns $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m \in \mathcal{H}$, an hyperplane in $\mathcal{H}$ is defined as $\{\mathbf{x} \in \mathcal{H} | \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$ where $\mathbf{w} \in \mathcal{H}$ and $b \in \mathbb{R}$, where vector $\mathbf{w}$ is orthogonal to the hyperplane. A canonical formulation of the hyperplane is given by the pair $(\mathbf{w}, b)$, for the set of samples $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m \in \mathcal{H}$ the closest training sample to it is

$$\min_{i=1,\cdots,m} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = 1,$$

which has a distance equivalent to $1/\|\mathbf{w}\|$.

The hyperplane separation margin and the magnitude of its orthogonal vector $\mathbf{w}$ play an important role on the algorithms based on support vectors. If all labelled training points are easily separated with a big margin there are reasons to believe that the same will happen while testing new patterns. A representative training set warranties the proximity of the testing samples to at least one of the trained samples. If all of them have a distance to the hyperplane bigger than ρ, small changes on the hyperplane parameters will not drastically modify the training classification results.

The pattern recognition problem using classifiers reduces to the problem of determining the function $f_{\mathbf{w},b} : \mathcal{H} \rightarrow \{\pm 1\}$ that maps or classifies correctly a labelled set $(\mathbf{x}_i, y_i) \in \mathcal{H} \times \{\pm 1\}$. This means that $f_{\mathbf{w},b} = y_i$ for every input pattern $\mathbf{x}_i$ (or a significant part of them). The objective here is not to obtain a classifier that makes the training errors disappear, it is much more important to estimate the parameters that *generalize* the information contained in the training set. This idea is illustrated in **Figure 2-1**, where three different margins are displayed for a simple 2D example with varying complexity. The model on the left (the less complex one) fails to capture the complexity expressed by the training samples misclassifying a large number of them. In contrast, the model on the right assigns to much trust to each point and offers a solution that is extremely complex and consistent with all the training points. The problem with this solution is that it may not work well with new unknown samples. This problem is popularly known as the *over-fitting* problem. The model in the middle shows

a solution that has a good classification rate for the training set and which is not affected by the presence of some atypical points (outliers).

The optimal separation hyperplane is constructed solving the following problem

$$\begin{aligned} &\min_{\mathbf{w}\in\mathcal{H}, b\in\mathbb{R}} \tau(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2, \\ &\text{subject to } y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq 1 \; \forall i \in 1, \cdots, m. \end{aligned} \tag{2.3}$$

Equation (2.3) is popularly known in the machine learning community as the *primal optimization problem* used to estimate the hyperplane parameters during the SVM formulation. Using the Lagrange multipliers[1], this same problem can be expressed in a more convenient form where the normal vectors **w** of the decision hyperplanes are represented as general linear combinations (*i.e.*, using non-uniform coefficients) of the training patterns. For this, the following Lagrangian equation is proposed

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i (y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1), \tag{2.4}$$

where the Lagrange multipliers are $\alpha_i \geq 0$. The Lagrangian equation in (2.4), is maximized with respect to $\boldsymbol{\alpha}$ and minimized with respect to **w** and b. The derivatives of L with respect to the primal variables at this saddle point are

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{2.5}$$

which leads us to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \text{ and } \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{2.6}$$

Equation (2.6) implies that the solution vector is an expansion expressed in terms of the training points. The patterns associated to $\alpha_i > 0$ correspond to the points that strictly satisfy

$$\alpha_i \left[y_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) - 1 \right] = 0, \tag{2.7}$$

they are called *support vectors*, and they are located exactly over the margin. All the other training samples are considered irrelevant, so that the influence of patterns located very far away from the decision boundary is removed.

2.1.1.1 Pedestrian detection with SVM classifiers

As with faces, pedestrians detectors are trained to identify regular 2D texture patterns that usually appear in upright people. As with the Viola and Jones dector (**Section 2.1.2)**, this patterns need to be sufficiently captured by an image sub-window representation. The histogram of oriented gradients method of Dalal and Triggs (**HOG**) [34] extracts a dense gradient-based descriptor from the window of interest. And an **SVM** (linear support vector machine) tells if it is a *person* or a *non-person*.

These descriptors need to be trained to separate positive samples (actual pedestrians) from the negative samples. **HOG** descriptors can be treated as points in an Euclidean vector space. Thus, **SVM** classification methods can be used directly

[1] The method of Lagrange multipliers (named after Joseph Louis Lagrange) is a strategy for finding the local maxima and minima of a function subject to equality constraints.

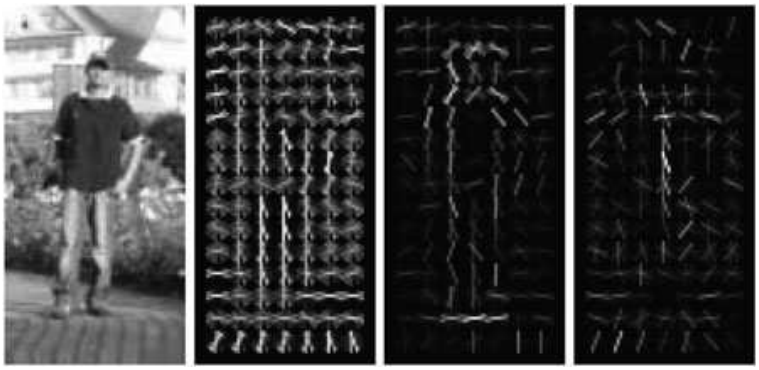


Figure 2-2: Linear **SVM** classifiers can help us to visualize the aspects of the **HOG** descriptor representation are distinctive. From a typical pedestrian window (extreme left) a **HOG** descriptor is computed (center left). Each of the orientation buckets in each window is a feature, a linear **SVM** assigns a weight to each feature. HOG features weighted by the positive weights are visualized in the center right image (important feature appear lighter). On the extreme right, weighted by the absolute value of negative weights is shown, highlighting the zones that strongly suggest a person is absent.

without any particular adaptation. During detection, an sliding window scans the whole images varying its location and scale. Multiple detections with slight variations in localization and scale can be found for each pedestrian, then *non-maximum* suppression is performed to filter or group all these detections into a single one.

As stated at the beginning of this section, pedestrians tend to take characteristic configurations: standing pedestrians have a wider upper body and narrower legs while walking pedestrians are bifurcated in the middle and show a scissors-like appearance. Linear **SVM** classifiers were chosen by Dalal and Triggs to identify these patterns from the **HOG** descriptors because of its simplicity (w.r.t. other classifiers) and performance. **SVM** classifiers highlight which features distinctively indicate the presence of an object from the ones which indicate its absence (see **Figure 2-2**).

2.1.2 AdaBoost classifier

The AdaBoost algorithm, introduced in 1995 by Freund and Schapire [47] takes an input as a training set $\{(x_1, y_1), \cdots, (x_m, y_m)\}$ where each x_i belongs to an instance space X, and each label y_i belongs to some label set Y and where m is the total number of learned samples. In the simplest case, the label set Y has only two possibilities $Y = \{-1, +1\}$ but it is possible to extend this algorithm to handle multi-class problems. AdaBoost calls a *weak* or *base learning algorithm* repeatedly, following a series of T rounds $t = 1, \cdots, T$ and maintaining a distribution of weights over the training set elements i (samples that are harder to classify receive a higher weight). Those weights are updated from round t to round $t + 1$. The weight assigned to the sample i at round t is denoted as $D_t(i)$. When the algorithm starts, the algorithm doesn't know which samples are going to be harder to classify, so, in the first round all weights may have the same value (*i.e.* $D_0(i) = 1/m$). At each round the accuracy of the classifier is evaluated, the weights $D_{t+1}(i)$ of incorrectly classified samples are increased forcing the next weak learner to focus on those *hard* samples that deserve more attention.

AdaBoost uses a linear combination of a collection of *weak classifiers* to construct a strong classifier of the form

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right). \tag{2.8}$$

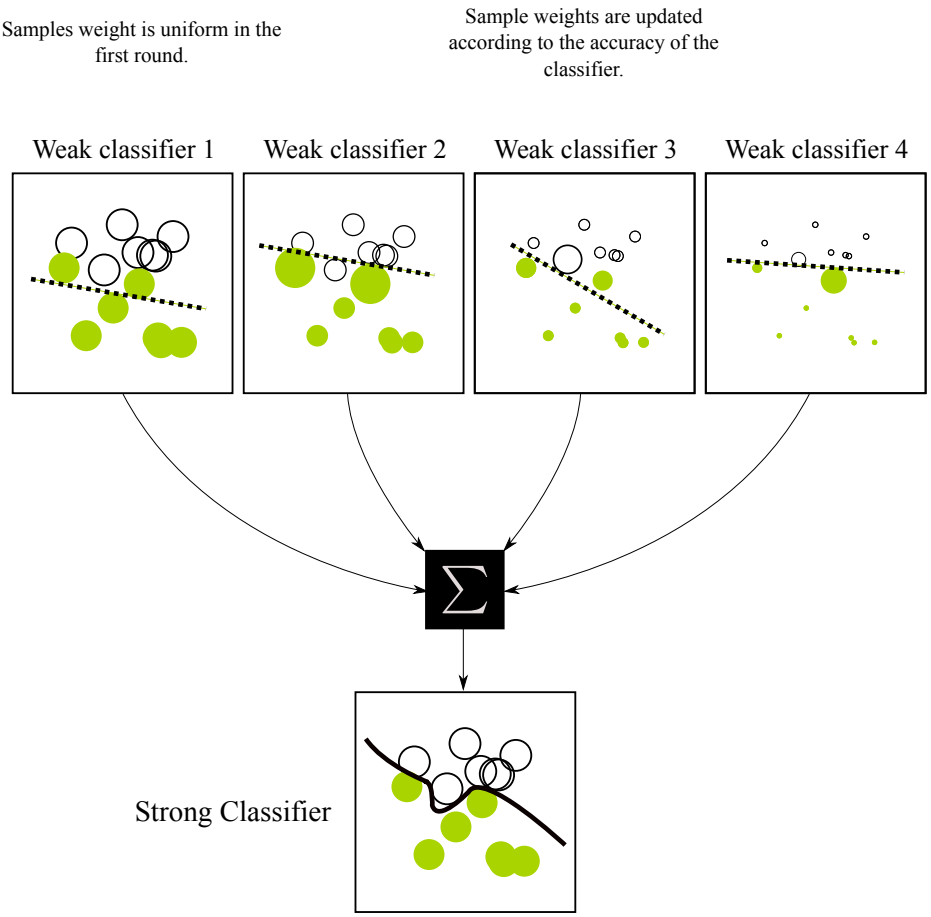


Figure 2-3: The linear combination of multiple *weak* classifiers responses forms a *strong* classifier.

During each round t the *weak hypothesis* h_t maps the samples in the input set X to the output set Y (*i.e.* $h_t : X \rightarrow \{-1, +1\}$). To measure how well this weak hypothesis fit the training data, the misclassified samples are penalized as follows

$$\epsilon_t = \Pr_{i \sim D_t} \left[h_t(x_i) \neq y_i \right] = \sum_{i : h_t(x_i) \neq y_i} D_t(i), \tag{2.9}$$

and h_t is the weak classifier that minimizes the classification error ϵ_t considering the weight distribution D_t.

In linear combination of the equation (2.8) each weak classifier is weighted by α_t. The value of α_t depends on the fitting error ϵ_t associated to the weak classifier h_t

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right), \tag{2.10}$$

the value of α_t is bigger as the fitting error ϵ_t decreases. The process stops if for any of the weak classifiers the error probability is $\epsilon_t \geq 0.5$ (more than 50% of the training samples are misclassified).

The complete AdaBoost training procedure is summarized in **Algorithm 8**. As the training advances, the values the next round $(t+1)$ on the weight distribution $D_{t+1}(i)$ are updated. Weights associated to the misclassified samples increase and those associated to the successfully classified samples go down (see the update equation in **Algorithm 8** line 6).

Algorithm 8: AdaBoost classification pseudo-code.

Data: List of labeled points $(x_1, y_1), \cdots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$.

Result: The final hypothesis: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$

Initialize $D_1(i) = 1/m$.

for $t = 1, \cdots, T$ **do**

Train the weak classifier using distribution D_t.

Get the weak hypothesis $h_t : X \to \{-1, +1\}$ and calculate the error $\epsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right]$.

Chose $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.

Update the weights using

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} \text{ if } h_t(x_i) = y_i \\ e^{\alpha_t} \text{ if } h_t(x_i) \neq y_i \end{cases}$$
$$= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}.$$

Z_t is a normalization factor that makes D_{t+1} a valid distribution.

Application example: Viola and Jones face detector

As an object category, frontal faces are well suited for window-based representations. Different face instances have a high regularity of 2D texture patterns. Viola and Jones use an sliding window alimenting generic and discriminative machine learning algorithm charged of discovering these regularities [144].

This face detection method is based on Haar wavelet features, and requires to compute pixel averages at multiple overlapping regions of different sizes both for training and during execution.

A large set of cropped image faces is first required for constructing the classifier, these images represent the *positive samples*. A large set of *negative samples* non-faces images is required too. Negative and positive samples are then re-scaled to a fixed resolution size *e.g.*, 24×24. A library of rectangular features windows is then constructed, each rectangular feature is a Haar-like filter which is parametrized by its relative location (inside the image sample), scale and type (see **Figure 2-4**). Where the output of a given filter applied to an image is the sum of the intensities of the pixels under the gray-shaded regions minus accumulation of pixels intensities under the white-shaded regions. Viola and Jones designate a bank of filters composed by nearly 180,000 rectangular features.

To avoid computing redundant accumulations multiple times, the rectangular features evaluation is accelerated using integral images as it was described in **Section 1.2.5**, this way, computing each rectangular feature response requires only 6-8 array references.

For an image $\mathbf{x}$, let $f_t(\mathbf{x})$ be the rectangular feature response and let θ_t be the threshold which minimizes the weighted

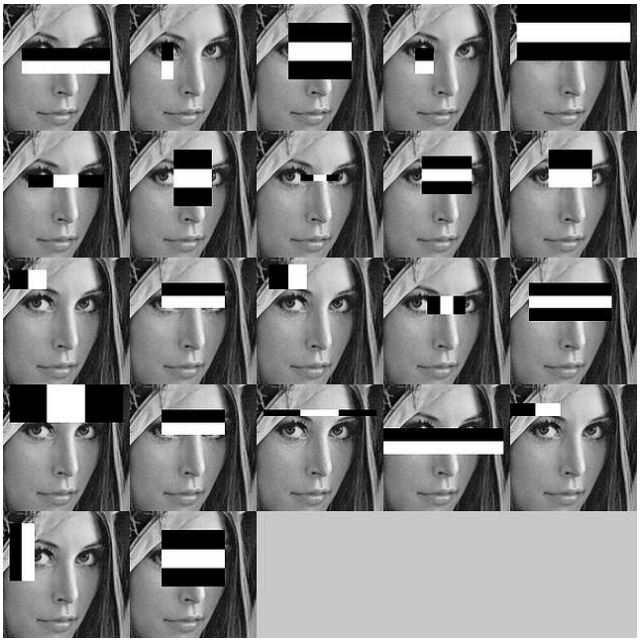


Figure 2-4: Some Haar-like rectangular features examples.

error on the training data at round t (for that feature). Thus, the *weak* classifier response is then

$$h_t(\mathbf{x}) = \begin{cases} +1 & \text{if } f_t(\mathbf{x}) > \theta_t \\ -1 & \text{otherwise,} \end{cases} \tag{2.11}$$

the tuple formed by the rectangular feature and its threshold θ_t define the learned feature.

A final *strong* classifier results from the weigthed addition of the individual *weak* classifiers after T rounds of boosting,

$$h(\mathbf{x}) = \begin{cases} +1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ -1 & \text{otherwise,} \end{cases} \tag{2.12}$$

where α_t is the classifier weight at round t. The value of $alpha_t$ is adjusted according to the error rate of its associated *weak* learner.

Given a test image and the ensemble of learned *weak* classifiers (2.11) together with their corresponding weights α_t that define the *strong* classifier the recognition process is very simple.

An sliding window traverses the whole image evaluating the *strong* classifier response (2.12) at a variety of locations and scales. During this evaluation, only the selected T feature responses are measured using the integral images method.

2.1.3 Classification on Riemannian Manifolds

Two different methods for classification in Riemannian manifolds are described in this section. The first approach is based on a collection of classifiers using trained at their tangent spaces and combined through boosting. A second option uses Kernel methods presented by Jayasumana *et al*. [69] to calculate the inner products in the Hilbert space $\mathcal{H}$ where the all the point x in the manifold $\mathcal{M}$ are mapped. But to respect the manifold structure, the kernel function must satisfy some properties described in **Section 2.1.3.3**.

2.1.3.1 LogitBoost for classification on Riemannian manifolds

Let $\{(\mathbf{X}_i, y_i)\}_{i=1,\cdots,N}$ be the training set with respect to the class labels, $\mathbf{X}_i \in \mathcal{M}$, $y_i \in \{0, 1\}$. The process of classifications maps each point $\mathbf{X}$ in the manifold as

$$F(\mathbf{X}) : \mathcal{M} \rightarrow \{0, 1\} \tag{2.13}$$

dividing the manifold into two parts based on the training set of labeled samples.

However, the idea of a function which divides the manifold is a complicated notion that is difficult to import from the Euclidean space. A straightforward (and naive) approach would be to map the manifold into a higher dimensional Euclidean space which can be interpreted as the flattening of the manifold. The problem is that there is no general mapping function that preserves the distances between all the points in the manifold which means that the global structure of the manifold is not respected by these mappings. Tuzel *et al*. deal with this situation using an incremental approach where several weak classifiers are trained at the tangent spaces and at the end they are combined through boosting. The matrix logarithm function at $\mathbf{X}$, $log_{\mathbf{X}}(\cdot)$ is employed (see equation (B.39) in the appendix for a definition of this function) to obtain the mapping of the neighborhood around $\mathbf{X} \in \mathcal{M}$ to the tangent space $T_{\mathbf{X}}$. Because this mapping is an homeomorphism, the local structure around the neighborhood of $\mathbf{X}$ in the manifold is preserved. And because the tangent space is an ordinary vector space, standard machine learning classifiers can be applied.

The Riemannian distance d_R between to points in the Riemannian manifold is approximated by the distance

$$d_R^2(\mathbf{Y}, \mathbf{Z}) \approx \|\text{vec}_{\mathbf{X}}(log_{\mathbf{X}}(\mathbf{Z})) - \text{vec}_{\mathbf{X}}(log_{\mathbf{X}}(\mathbf{Y}))\| \tag{2.14}$$

and the approximation error can be expressed in terms of the sum of the pairwise distances

$$\varepsilon = \sum_{i=1}^{N}\sum_{j=1}^{N}(d_R(\mathbf{X}_i, \mathbf{X}_j) - \|\text{vec}_{\mathbf{X}}(log_{\mathbf{X}}(\mathbf{X}_i)) - \text{vec}_{\mathbf{X}}(log_{\mathbf{X}}(\mathbf{X}_j))\|). \tag{2.15}$$

The best point in the manifold to train the classifier is the one which provides the best approximation or the minimum error in equation (2.15). It has been determined empirically that the mean of the training points is a good minimizer. Weak learners are then trained on the tangent space at the mean of the training points, at each iteration, a new weighted mean is computed (the weights are adjusted by boosting) and the miss-classified samples receive greater weights. This way, the mean tends to move towards these points and more accurate classifiers are then learned.

The classification procedure used in [140] is the LogitBoost algorithm [48] which operates on vector spaces. Here, the

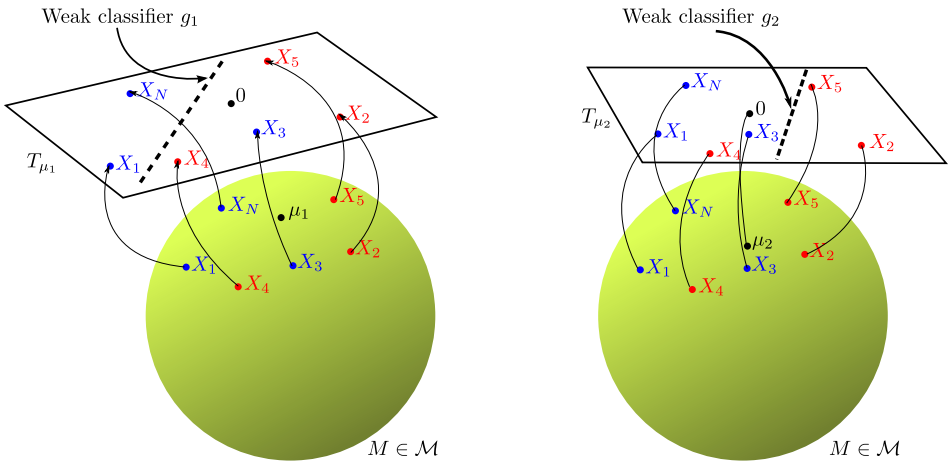


Figure 2-5: Classification in Riemannian manifolds example. Here the manifold $M \in \mathcal{M}$ is depicted as the surface of the sphere and the tangent spaces are represented by the planes. The training samples are projected to the tangent space via log_{μ_l}, weak classifiers are learned on the tangent spaces T_{μ_l}. Sample $\mathbf{X}_3$ is misclassified for the weak learner on the left g_1, its weight increases pulling the mean μ_2 in the second iteration towards it.

probability of $\mathbf{x}$ being in class 1 is given by

$$p(\mathbf{x}) = \frac{e^{F(\mathbf{x})}}{e^{F(\mathbf{x})} + e^{-F(\mathbf{x})}}, \qquad F(\mathbf{x}) = \frac{1}{2}\sum_{l=1}^{L} f_l(\mathbf{x}) \tag{2.16}$$

which depends on the set of weak learners regression functions $\{f_l(\mathbf{x})\}_{l=1\cdots L}$ that minimize the negative binomial log likelihood of the data

$$-\sum_{i=1}^{N}\left[y_i \log(p(\mathbf{x}_i)) + (1-y_i)\log(1-p(\mathbf{x}_i))\right], \tag{2.17}$$

fitting a weighted least squares regression $f_l(\mathbf{x})$ of the training points $\mathbf{x}_i \in \mathbb{R}^m$ to the response values z_i with the associated weights w_i, where

$$z_i = \frac{y_i - p(\mathbf{x}_i)}{p(\mathbf{x}_i)(1-p(\mathbf{x}_i))}, \qquad \text{and } w_i = p(\mathbf{x}_i)(1-p(\mathbf{x}_i)). \tag{2.18}$$

The Riemannian version of the Logit-boost classification method is described completely in **Algorithm 9**.

Algorithm 9: LogitBoost classification on Riemannian manifolds

Data: A labeled training set $\{(\mathbf{X}_i, y_i\}_{i=1,\cdots,N}, \mathbf{X}_i \in \mathbb{M}, y_i \in \{0,1\}$

Start with uniform weight values $w_i = \frac{1}{N}$, $i = 1, \cdots, N$, $F(\mathbf{X}) = 0$ and $p(\mathbf{X}_i) = \frac{1}{2}$

for $l = 1 \cdots L$ **do**

  Compute response values and weights: $z_i = \frac{y_i - p(\mathbf{X}_i)}{p(\mathbf{X}_i)(1-p(\mathbf{X}_i))}, w_i = p(\mathbf{X}_i)(1-p(\mathbf{X}_i))$

  Compute the wighted mean of the points (Riemannian Covariance Mean): $\mu_l = \arg\min_{\mathbf{X}\in\mathcal{M}} \sum_{i=1}^{N} w_i d^2(\mathbf{X}_i, \mathbf{X})$

  Map the data point to the tangent space at μ_l: $\mathbf{x}_i = vec_{\mu_l}(\log_{\mu_l}(X_i))$

  Fit the function $g_l(\mathbf{x})$ by the weighted least-squares regression of z_i to $\mathbf{x}_i$ using weights w_i.

  $F(\mathbf{X}) \leftarrow F(\mathbf{X}) + \frac{1}{2} f_l(\mathbf{X})$, where $f_l = g_l(vec_{\mu_l}(\log_{\mu_l}(\mathbf{X}))$ and $p(\mathbf{X}) \leftarrow \frac{e^{F(\mathbf{X})}}{e^{F(\mathbf{X})}+e^{-F(\mathbf{X})}}$

Store $F = \{\mu_l, g_l\}_{l=1\cdots L}$

Output the classifier sign $(F(\mathbf{X})) = \text{sign}\left(\sum_{l=1}^{L} f_l(\mathbf{X})\right)$

2.1.3.2 Pedestrian classification on Riemannian manifolds with LogitBoost classifiers

Tuzel *et al*. presented in [140] a method for detecting pedestrians using classification methods in the Riemannian manifold where *MSE* matrices such as covariance matrices lie.

The proposed feature vector in [140] for pedestrian detection is

$$\phi(I,x,y) = \left[\begin{array}{cccccccc} x & y & |I_x| & |I_y| & \sqrt{I_x^2+I_y^2} & |I_{xx}| & |I_{yy}| & \arctan\frac{|I_x|}{|I_y|} \end{array} \right]^T, \tag{2.19}$$

where (x,y) define the pixel locations, I_x,I_{xx},$\cdots$ are the intensity derivatives and the the arctan $\frac{|I_x|}{|I_y|}$ function defines the orientation of the edge. This 8-dimensional feature vector results in a 8×8 covariance matrix descriptor which encodes the variances of the features, their correlations and their spatial layout. The algorithm receives an arbitrary sized detection window R where there is a very large number of different and overlapping subregions $r_{1,2,\ldots}$ where covariance descriptors can be computed. To explore them, the algorithm starts with the regions of $\frac{1}{10}$ the width and height of the detection window R. After exploring each scale, the size is then incremented in steps of $\frac{1}{10}$, until $r = R$.

Tuzel *et al*. in [140] used a cascade of $K = 30$ LogitBoot classifiers operating on 8×8 *SPD* matrices. The linear regression functions (learners) $g_{k,l}$ are located in a 36-dimensional vector space (the upper triangle of the 8×8 covariance matrix). During classification, the algorithm receives a set of labeled pairs $\{(R_i, y, i)\}_{i=1\cdots N}$, where R_i are the image windows containing background and pedestrians and $y_i \in 0, 1$ are the labels. As explained before, for each detection window R there is a very large number of covariance descriptors which can be computed, so, each weak classifier is associated to one of these sub-windows in R. The sub-window associated to the l-th classifier of cascade level k is denoted by $r_{k,l}$. While training in the k-level, the algorithm classifies all the negative examples $\{R_i^-\}_{i=1,\cdots,N_n}$ (N_n is the total number of negative samples) with the previous $(k-1)$ LogitBoost levels. All the correctly classified samples are removed from the training set (all sub-windows from negative images are negative). Each one of the K LogitBoost classifiers can have a different number of learners L_k. The idea is to optimize each cascade level until 99.8% of the positive samples are detected and at least 35% of the negative samples are rejected. At each iteration l of the k-th LogitBoost cascade level, 200 sub-windows are sampled among all the possible sub-windows. Weak classifiers are learned representing each sub-window and the classifier that best minimizes the log likelihood function (2.17) is appended to list of classifiers of level k. The same procedure is repeated until the specified detection rates are satisfied.

2.1.3.3 Kernel methods on Riemannian manifolds

Kernel methods in $\mathbb{R}^n$ are extremely effective in machine learning and computer vision to explore non-linear patterns of data. Kernel methods map the input data into a high dimensional feature space where a richer representation of the data distribution is attainable. With kernel methods, each point x on a non-linear manifold $\mathcal{M}$ is mapped to a feature vector $\phi(x)$ in a Hilbert space[2] $\mathcal{H}$. A kernel function $k : (\mathcal{H} \times \mathcal{H}) \rightarrow \mathbb{R}$ is used as the inner product on $\mathcal{H}$. It should be noted that in general, Riemannian manifolds are non-linear and that many algorithms designed for $\mathbb{R}^n$ are not available for them. The method described in **Section 2.1.3.1** maps the points on the manifold to the tangent space at one particular point (usually the

[2] A Hilbert space $\mathcal{H}$ is a real or complex inner product space that is also a complete metric space with respect to the distance function induced by the inner product.

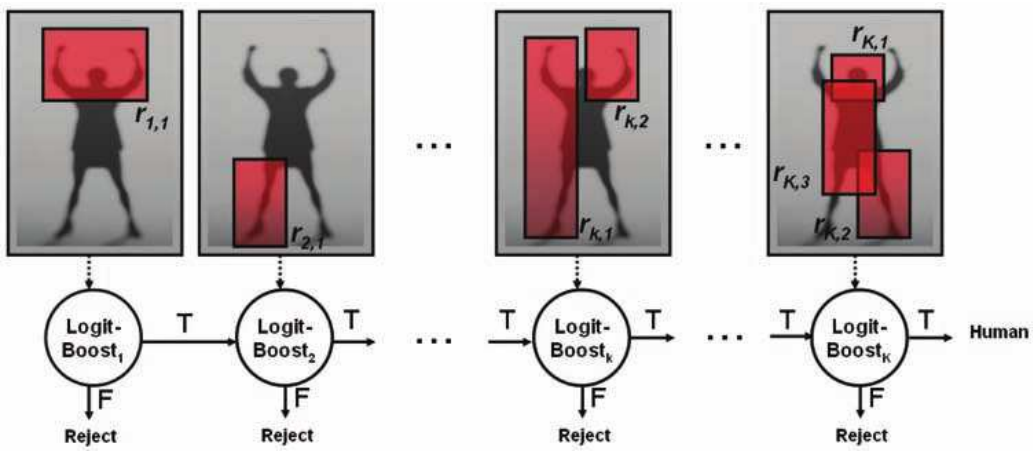


Figure 2-6: Cascade of LogitBoost Riemannian classifiers for human detection, each classifier k in the cascade has a variable number of classifiers L_k a detection is declared when the a sample passes through all the classifiers.

mean point) obtaining a local Euclidean representation. Unfortunately, that idea does not globally preserve point distances and produces a poor representation of the manifold's structure (data distribution). Mapping transformations of the non-linear manifold into a Hilbert space make possible to utilize algorithms designed for $\mathbb{R}^n$ with manifold valued input data. However, these benefits are obtained when the selected kernel is positive definite only.

Jayasumana *et al*. proposed in [69] a radial basis function (RBF) that maps the input data to an infinite dimensional Hilbert space in $\mathbb{R}^n$. The Gaussian kernel $k_G(\mathbf{x}_i, \mathbf{x}_j) := \exp\left(\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2\right)$ based on the Euclidean distance between to data points $\mathbf{x}_i$ and $\mathbf{x}_j$ is used. A kernel on a Riemannian manifold can be obtained by replacing the Euclidean distance by a geodesic on the manifold. However, not every geodesic distance yields a positive definite kernel. The sufficient and necessary conditions for a geodesic metric to generate a valid positive definite kernel are not described here but can be found in [69].

The log-Euclidean distance for Sym_d^+ (SPD matrices) is derived by exploiting the Lie group structure under the group operation $\mathbf{X}_i \odot \mathbf{X}_j := \exp\left(\log(\mathbf{X}_i) + \log(\mathbf{X}_j)\right)$. Here, the $\exp(\cdot)$ and $\log(\cdot)$ operators denote the usual matrix exponential and logarithm operators and not the point dependent exponential and logarithm maps of the log-Euclidean Riemannian metric. A geodesic which connects the points $\mathbf{X}_i, \mathbf{X}_j \in Sym_d^+$ is parametrized as $\gamma(t) = \exp\left((1-t)\log(\mathbf{X}_i) + t\log(\mathbf{X}_j)\right)$ for $t \in [0, 1]$. The geodesic distance between these points is thus expressed as

$$d_g(\mathbf{X}_i, \mathbf{X}_j) = \|\log(\mathbf{X}_i) - \log(\mathbf{X}_j)\|_F, \tag{2.20}$$

where $\|\cdot\|_F$ denotes the Frobenius matrix norm[3]. The log-Euclidean distance defines a true geodesic distance measure on Sym_d^+ and produces a valid positive definite kernel.

Being able to compute positive definite kernels allows us to use algorithms developed for $\mathbb{R}^n$ while respecting the manifold's geometry. This way, the problem of sample classification on the Sym_d^+ manifold using Kernel SVM's given the set of training samples $\{(\mathbf{X}_i, y_i)\}_1^m$, where $\mathbf{X}_i \in Sym_d^+$ and the label $y_i \in \{\pm 1\}$ turns out in the research of the hyperplane in $\mathcal{H}$ that optimally separates the feature vectors $\phi(\mathbf{X}_i)$ that belong to the positive and negative classes with a

[3]Defined in various ways such as $\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2} = \sqrt{\text{trace}(A^* A)} = \sqrt{\sum_{i=1}^{\min\{m,\, n\}} \sigma_i^2}$.

maximum margin. The relative position of the point $\phi(\mathbf{X}_i) \in \mathcal{H}$ to the hyperplane is what determines the class to which the sample belongs to. Other kernel-based algorithms can be used similarly on the manifold of Sym_d^+ matrices such as multiple kernel learning (MKL), kernel principal component analysis (PCA) or kernel k-means clustering.

2.2 Object matching

Many different matching methods have been proposed by the computer vision community. Four matching methods which inspired this thesis are presented here.

- **Template matching**: comparing image patches directly using simple measures such as the **mean squared error** (*MSE*), **normalized cross-correlation** (NCC) or the **structural similarity index** (SSIM). These approaches are more suitable for to track small patches and small rigid planar objects.
- **Median KLT optical flow**: popularly known as the *flock of trackers* algorithm, this method is able to track non-rigid objects analyzing the KLT optical flow and using the median operator to estimate the moving direction of a target composed by multiple local patches.
- **Mean-Shift**: In Mean-Shift MS tracking [30], the target is modeled by a color-space representation, and the tracking is achieved by a gradient-based optimization using the Bhattacharyya distance.
- **Covariance tracking**: Using the covariance descriptor for matching it is possible to perform local and exhaustive searches. Some methods that incorporate information about the target dynamics to estimate the most plausible displacements are mentioned too.

2.2.1 Template matching by correlative methods

The **mean squared error** (*MSE*) is a popular measure of fidelity which is also interpreted as the degree of similarity or, conversely, the level of error and distortion between two different signals. One of these two signals is a template representing a target and the other is the candidate location. Consider a template image $T(\mathbf{x}_i)$ defined by the N pixels $\mathbf{x}_i \forall i \in \{0, 2, \cdots, N-1\}$ inside of it. A patch of the same size $P(\mathbf{x}_i)$ is taken from the target image at the candidate's location. The *MSE* between these two images is

$$MSE(T, P) = \frac{1}{N} \sum_{i=0}^{N-1} (T(\mathbf{x}_i) - P(\mathbf{x}_i))^2. \tag{2.21}$$

We can also refer to the error signal $e_i = T(\mathbf{x}_i) - P(\mathbf{x}_i)$, which is the difference between the template and the candidate images. A more general form would be the ℓ_p norm

$$d_p(T, P) = \left(\sum_{i=0}^{N-1} |e_i|^p \right)^{1/p}. \tag{2.22}$$

MSE is often converted into a peak signal-to-noise ratio measure (*PSNR*).

MSE has many attractive features: 1)it is simple and inexpensive to compute, 2) all ℓ_p norms are valid distance metrics in $\mathbb{R}$ satisfying all the conditions of non-negativity, identity, symmetry and triangular inequality, 3) *MSE* measures the energy of the error signal e_i which has a clear physical meaning, and finally, 4) *MSE* is suitable for optimization problems such as gradient-descent and Hessian matrix (as seen for the *KLT* optical flow in **Section 1.1.5**) allowing to search locally for the minimum-*MSE*. On the other hand, template matching with *MSE* has some severe inconveniences: 1) all signal samples are equally important, 2) *MSE* is independent of the signs of the error e_i and 3)temporal or spatial relationships between pixels are not considered.

Normalized cross-correlation (*NCC*) represents a different approach for measuring the degree of similarity between an image and a template. One advantage of *NCC* over *MSE* is that it is usable for conditions in which the brightness of the image and the template significantly vary. The equation that defines *NCC* is

$$NCC = \frac{1}{N}\sum_{i=0}^{N-1}\frac{(T(\mathbf{x}_i)-\overline{T})(P(\mathbf{x}_i)-\overline{P})}{\sigma_T\sigma_P}, \tag{2.23}$$

where $\overline{T}$ and $\overline{P}$ are the averages of the template and the candidate's patch and σ_T and σ_P are their standard deviations. *NCC* can be interpreted as the dot product of two normalized vectors. Similar to *MSE*, *NCC* imposes significant limitations on the signal samples. Both are unable to view the interaction of samples between each other or how the image structures are affected by and additive error. The **structural similarity index** (*SSIM*), have proven being useful as an image fidelity measurement. *SSIM* can be implemented at a single scale, over multiple scales or in the wavelet domain. The approach followed by *SSIM* is motivated on the observation that natural image signals are highly structured, which means that image signals often have strong neighbor dependencies that carry important information about the structures of the objects. Supposing that T and P are the template and the local patch we want to compare, *SSIM* measures the similarity of three different elements of them: the similarity of their local luminances $l(T,P)$, the similarity of the local patch contrasts $c(T,P)$, and the similarity of the local patch structures $s(T,P)$. All these similarities are obtained using simple local statistics as

$$SSIM(T,P) = \underbrace{\left(\frac{2\mu_T\mu_P+C_1}{\mu_T^2+\mu_P^2+C_1}\right)}_{l(T,P)}\cdot\underbrace{\left(\frac{2\sigma_T\sigma_P+C_2}{\sigma_T^2+\sigma_P^2+C_2}\right)}_{c(T,P)}\cdot\underbrace{\left(\frac{\sigma_{TP}+C_3}{\sigma_T\sigma_P+C_3}\right)}_{s(T,P)}, \tag{2.24}$$

where μ_T and μ_P are the sample means of T and P, σ_T and σ_P represent their standard deviations and σ_{TP} is the sample cross correlation of T and P after removing their means. The terms C_1, C_2 and C_3 are small positive constants used to stabilize each of the similarity terms so that sample means, variances or correlations do not lead to numerical instabilities when they are close to zero. *SSIM* has nice properties such as symmetry ($S(T,P)=S(P,T)$), it is also bounded: $-1 < S(T,P) \leq 1$ (achieving the maximum value $S(T,P)=1$ when both images are the same). *SSIM* index can be computed locally with a sliding window that moves pixel-by-pixel across the image mapping them to a *SSIM* similarity image. A *SSIM* score can be obtained averaging over this image.

2.2.2 Median flow (Flock of Trackers) object tracking

In **Section 1.1.5**, the principles Lucas-Tomasi-Kanade algorithm (*KLT*) [8] were reviewed. *KLT* receives a list of key-point locations at time t and obtains their location at time $t+1$ by estimating the linear transformation that makes the local patches approximately fit. If a target is regarded as a simple collection (a cloud) of feature points, the estimation of the optical flow on the key-points associated to it should give a valuable cue to estimate the target overall state (*e.g.*, pose, scale, location). The problem in practice is that optical flow algorithms like *KLT* face significant challenges such as dramatical changes in key-point appearance, key-point disappearances and occlusions resulting in tracking failures. Reliable target tracking algorithms need to be able to estimate which points deserve more confidence than the rest and which points are outliers. A surrounding patch $\mathcal{W}$ is commonly assumed, patches taken at times t and $t+1$ are compared using the mean squared error (*MSE*), normalized-cross-correlation (*NCC*) or the structural similarity *SSIM*. These measures methods are useful to detect occlusions and rapid movements in planar objects, but they may fail for non-rigid objects and slowly drifting trajectories.

Kalal *et al*. proposed in [73] a forward-backward error used the detect optical-flow failures. The principle behind it is very simple: tracking should be independent of the direction of time-flow. A key-point traces a trajectory forward in time, then at the last frame of the sequence this trajectory is validating tracking the same point backward in time. If both trajectories significantly differ, the point is considered unreliable. Forward-backward is consuming because it needs to track two times a feature (once for each time sense), some alternative methods have been proposed to estimate the consistency of the trajectories. Key-points are usually part of bigger units (cars, pedestrian, hands, faces, etc.) that move in groups with a certain degree of coherence. In [146], Wendel *et al*. use the trackers neighborhood consistency and a temporal predictor (Markov chain) to evaluate how much confidence the algorithm assigns on them.

When a new target is detected, as set of points is initialized forming a rectangular grid within the target bounding box. Each point is then tracked by *KLT* generating an sparse motion flow between images I_t and I_{t+1}. Each point flow is then analyzed and evaluated using the aforementioned methods, 50% of the worst (less confident) trackers are then filtered out. The remaining points are then used to estimate the displacement of the whole target using the median over each spatial dimension, this is the reason why this method is referred as *Median Flow* (*MF*) or *Flock-of-Trackers* (*FoT*). From now on, the name used in this work to refer to this method will be *FoT*.

For each pair of points, a ratio is formed using their distance in the current t and previous $t-1$ frames. The median of all ratios provides an estimation the target's change of size which depends on the optical flow results only.

Thanks to the robustness of the median, *FoT* can resist up to 50% of outliers in the translation estimation and $100 \times (1 - \sqrt{0.5})\%$ for the estimation of the scale change. Tolerance to outliers can be even higher because outliers do not *conspire* and in most cases they are distributed evenly above and below the median. Nevertheless, it is still possible to find scenarios where the inlier percentage is extremely low and where the translation and scale change estimation are very difficult. In the original *FoT* version, all the trackers are placed regularly over the object forming a grid, it is clear that not all these trackers will be placed at good locations suitable for tracking, and these poorly placed trackers tend to drift away from their original position on the grid and behave as outliers. After estimating the global target displacement and scale change, all these targets are reseted to their original place on the grid. Arguing that this approach is suboptimal, Wendel *et al*. replace the grid by a set of *cells*. In this version of the *FoT* method, trackers are allowed to move and *find* their suitable offset within their cell *i.e.* their best position for tracking. Cells guarantee an even coverage of the target, forcing the trackers to

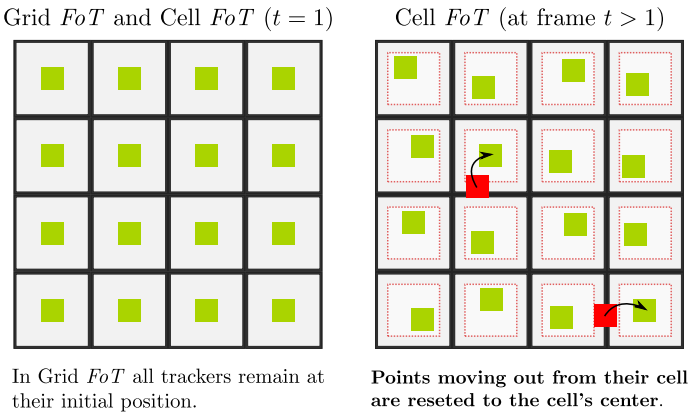


Figure 2-7: Grid FoT and Cell FoT comparison. In both methods all the trackers are initiated at their corresponding positions in the grid, Cell FoT lets the trackers move within their cells, but they are relocated when they get out from their cells.

stay inside inside of them. If a tracker moves out from its cell, it is repositioned at the cell's center.

2.2.3 Mean-Shift (MS tracking)

In MS tracking [30], the target is modeled by a color-space representation, and the tracking is achieved by a gradient-based optimization. The similarity between the target model at initial location $\mathbf{x}_0$ and the target candidate at location $\mathbf{x}_t$, is computed as the Bhattacharyya distance, noted ρ, and based on the bin-to-bin product of their respective color distributions.

If a color appears on both the object and its vicinity or background, it is not relevant for tracking because it reduces their separability. A lower confidence has to be granted to that color, as in [5], where the background colors are subtracted from the histogram using the log-likelihood ratio of foreground/background.

The histogram is generally quantized in order to allow real-time execution and avoid sparsity. Consequently, after background subtraction and quantization, the histogram can be close to empty, and the similarity measure might vanish even for small changes in the color distribution.

Considering the reference target model $\widehat{\mathbf{q}}_u$, the tracking consists in finding in frame t the candidate location $\mathbf{x}_t$ for which the representation $\widehat{\mathbf{p}}_u^t(\mathbf{x}_t)$ maximizes the similarity to the model (u is the index on the bins). The Bhattacharyya distance is expanded in Taylor series as in [30] in order to allow gradient-based optimization. Iteratively, each pixel of index i and color $\mathbf{c}_i$ in the target contributes to the computation of the new location, with a weight w_i, defined as follows:

$$w_i = \sum_u \sqrt{\frac{\widehat{\mathbf{q}}_u}{\widehat{\mathbf{p}}_u^t(\mathbf{x}_t)}}\delta(\mathbf{c}_i - u), \tag{2.25}$$

with δ being the Kronecker function. The success of MS relies on the good description of the target by its color distribution. Ideally, the color-space has to be chosen so that: 1) the histogram is not empty after background subtraction; 2) the weights (2.25) are non-zero on the target and are numerous enough.

2.2.4 Covariance descriptor tracking

Given an image representing the target, the aim of covariance tracking is to locate the object and its pose even after non-rigid transformations in an arbitrary image. The original combination of features proposed by Tuzel *et al*. [139] considers pixel locations (x, y), color values (RGB components) and the norm of the first and second order derivatives of the intensities with respect to x and y. After the initial mapping, each pixel is converted to a nine-dimensional feature vector

$$F(x,y) = \left[\begin{array}{ccccccc} x & y & R(x,y) & G(x,y) & B(x,y) & \left|\frac{\partial I(x,y)}{\partial x}\right| & \left|\frac{\partial I(x,y)}{\partial y}\right| & \left|\frac{\partial^2 I(x,y)}{\partial x^2}\right| & \left|\frac{\partial^2 I(x,y)}{\partial y^2}\right| \end{array} \right]^T, \tag{2.26}$$

thus, the resulting covariance region descriptor is a 9×9 matrix.

Tracking algorithms search in the target image for the region that has the most similar covariance matrix, the dissimilarity expresion is presented in this subsection. The simplest tracking algorithm initially proposed by Tuzel *et al*. scans the target image using a brute force approach, testing different locations and scales. This approach is suitable for single object tracking applications where there is no available information about the target's dynamics and where unpredictable changes on the location and scale can occur from one image to the next. Probabilistic search methods take the set of N previous locations $\{\mathbf{x}_{t-(N-1)}, \ldots, \mathbf{x}_t\}$ and their covariance descriptors to fit the transition functions representing the evolution of the target's dynamics (*e.g*., translation, rotation, scale, shear).

2.2.4.1 Distance calculation: Riemannian metric for SPD matrices

Covariance models and instances can be compared and matched using a simple nearest neighbor approach (*i.e*., finding the covariance descriptors that best resembles a model). The problem is that covariance matrices (*SPD* matrices in general) do not lie on the Euclidean space and many common and widely known operations in Euclidean spaces are not applicable or require to be adapted (*e.g*., a *SPD* matrix multiplied by a negative scalar is no longer a valid *SPD* matrix). A $n \times n$ *SPD* matrix only has $n \times (n+1)/2$ different elements, while it is possible to vectorize them and perform element-by-element subtraction, this approach provides very poor results as it fails to analyze the correlations between variables and the patterns stored in them. A solution to this problem is proposed in [44] where a dissimilarity measure between two covariance matrices is given as

$$\rho(\mathbf{C}_1, \mathbf{C}_2) = \sqrt{\sum_{i=1}^{n} \ln^2 \lambda_i(\mathbf{C}_1, \mathbf{C}_2)} \tag{2.27}$$

where $\{\lambda_i(\mathbf{C}_1, \mathbf{C}_2)\}_{i=1,\cdots,n}$ are the generalized eigenvalues of $\mathbf{C}_1$ and $\mathbf{C}_2$ computed from

$$\lambda_i \mathbf{C}_1 \mathbf{x}_i - \mathbf{C}_2 \mathbf{x}_i = 0 \qquad i = 1, \cdots, d. \tag{2.28}$$

In (2.28), $\mathbf{x}_i \neq 0$ are the generalized eigenvectors. Distance measure (2.27) satisfies the metric axioms for *SPD* matrices $\mathbf{C}_1$ and $\mathbf{C}_2$

$$\begin{aligned} &1.\ \rho(\mathbf{C}_1, \mathbf{C}_2) \geq 0 \text{ and } \rho(\mathbf{C}_1, \mathbf{C}_2) = 0 \text{ only if } \mathbf{C}_1 = \mathbf{C}_2, \\ &2.\ \rho(\mathbf{C}_1, \mathbf{C}_2) = \rho(\mathbf{C}_2, \mathbf{C}_1) \\ &3.\ \rho(\mathbf{C}_1, \mathbf{C}_2) + \rho(\mathbf{C}_1, \mathbf{C}_3) \geq \rho(\mathbf{C}_2, \mathbf{C}_3). \end{aligned} \tag{2.29}$$

2.2.4.2 Exhaustive search covariance tracking

Computing the covariance descriptors in the brute force search mechanism is not expensive due to the integral images method discussed in **Section 1.2.5**. Without this method computing these descriptors for multiple arbitrary sized and overlapping regions over the whole image would be extremely prohibitive for real-time implementations.

The algorithm receives the input image I and applies to it the feature map $\phi(x, y)$ obtaining the set of N feature images. Integral images are calculated for these N feature images (*primary features*) and their $N(N+1)/2$ crossed-products (*second order features*). This will allows us to calculate feature averages over arbitrary sized regions very fast. Applying an sliding window and varying its size, the algorithm computes the candidate's covariance descriptors and compares them against the template model using the Riemannian metric

$$d^2(\mathbf{M}, \mathbf{Y}) = \text{tr}\left[\log^2(\mathbf{M}^{-\frac{1}{2}}\mathbf{Y}\mathbf{M}^{-\frac{1}{2}})\right], \tag{2.30}$$

the denotes the distance between the sample $\mathbf{Y}$ and the covariance model $\mathbf{M}$.

The most likely location for the target in I is the region that has the covariance descriptor which is closest to the template's model. This method works reasonably well for single target tracking algorithms or image retrieval applications but not for multiple target tracking. The reason is that this method imposes no restriction on the targets dynamics and trackers of similar appearance get confused easily which results in identity swaps that may occur if a mechanism to prevent them is absent.

2.2.4.3 Probabilistic search methods

Wu *et al*. propose to regard tracking as a probabilistic inference of the target state [149]. Having the set $\{\mathbf{x}_i\}_{i=0:t}$ of target states at time $0, \cdots, t$ and the observation $\{\mathbf{y}\}_{0:t}$, the method constructs a probabilistic function $p(x_t|y_{0:t})$. Where the target density propagation is expressed as

$$p(\mathbf{x}_t|\mathbf{y}_{0:t}) \propto p(\mathbf{y}_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{0:t-1}) d\mathbf{x}_{t-1}. \tag{2.31}$$

In a sequential Monte Carlo framework, the posterior $p(\mathbf{x}_t|\mathbf{y}_{0:t})$ is approximated by the weighted sample set $\{\mathbf{x}_t^n, w_t^n\}_{n=1}^{N_s}$, where $\sum_{n=1}^{N_s} w_t^n = 1$ and where all the N_s particles are sampled from the density $q(\mathbf{x}_t^n|\mathbf{x}_{t-1}^n, \mathbf{y}_t)$. The weight associated to each particle is

$$w_t^n \propto \frac{p(\mathbf{y}_t|\mathbf{x}_t^n) p(\mathbf{x}_t^n|\mathbf{x}_{t-1}^n)}{q(\mathbf{x}_t^n|\mathbf{x}_{t-1}^n, \mathbf{y}_t)} w_{t-1}^n. \tag{2.32}$$

From time to time, all the particles are re-sampled and their weights are re-started to $\frac{1}{N_s}$. Then, the Monte Carlo approximation of the expectation $\hat{\mathbf{x}}_t = \frac{1}{N_s} \sum_{n=1}^{N_s} w_t x_t^n \approx E(\mathbf{x}_t|\mathbf{y}_{0:t})$, is used to estimate the state $\mathbf{x}_t$. An example of the Monte Carlo approach for tracking with covariance matrices is shown in **Figure 2-8**.

The target state $\mathbf{x} = (\mathbf{d}, \mathbf{s}, \mathbf{v})$ stores information about the location $\mathbf{d} = (x, y)$, the scale $\mathbf{s} = (w, h)$ and its speed $\mathbf{v} = (v_x, v_y)$. It is common to model state transitions with a first-order ($B = 0$) or second-order auto-regressive dynamic model

$$\mathbf{x}_t = A\mathbf{x}_{t-1} + B\mathbf{x}_{t-2} + C\mathcal{N}(0, \Sigma), \tag{2.33}$$

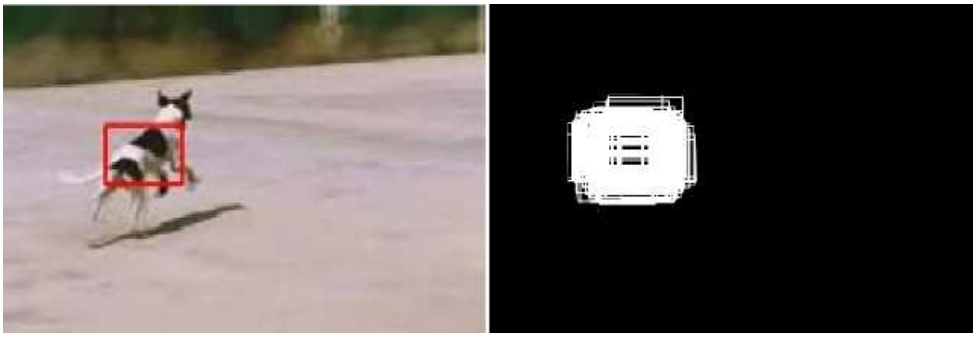


Figure 2-8: Covariance tracking with a Monte Carlo framework, the state of the target at time $t-1$ is shown with a red rectangle, a set of N particles is sampled near tho this location (shown in white) and are used to estimate the target transition from state $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$.

but matrices A, B, C and Σ that define the target dynamics are difficult to estimate and [149] propose a sampling scheme to model the transitions as

$$\begin{aligned} d_t^n &= d_{t-1}, v_t^n = 0 \quad u < u_0 \\ d_t^n &= d_{t-1}^n + v_{t-1}^n \quad u \geq u_0, \end{aligned} \tag{2.34}$$

where u is a random number from the distribution $U(0,1)$, $u_0 \in [0,1]$, d_t^n is the location of the n-th particle in time t, d_{t-1} is the location of the target at $t-1$, v_t^n and v_{t-1}^n are the velocity of the n-th particle in time t and $t-1$ respectively.

2.2.4.4 Steepest descent covariance tracking

The search methods discussed in **Section 2.2.4.3** are sub-optimal both in terms of accuracy and execution time. The reason is that global searches find the best match by comparing exhaustively in the whole image. This problem is more evident in high-resolution images or big-sized objects. Moreover, the likelihood of finding distractions (objects with similar covariance descriptors) is greater when the whole image is scanned without considering the most probable target locations. As a workaround, Tyagi *et al*. propose a gradient descent based method [141] expecting that the target's location changes gradually (the prior of finding the object is higher nearby the previously known location).

Avoiding the computation of distance metrics in numerous locations, the steepest descent method of [141] saves computational resources. The gradient descent method for covariance matrices can be formulated as an optimization problem that only requires the calculation of the local gradient at the current location.

Considering the Riemannian metric of equation (2.30), it is possible to state the problem in terms of the current location $\mathbf{x}$ as

$$f(\mathbf{x}) = d^2(\mathbf{M}, \mathbf{Y}_\mathbf{x}) = \text{tr}\left[\log^2\left(\mathbf{M}^{-\frac{1}{2}}\mathbf{Y}_\mathbf{x}\mathbf{M}^{-\frac{1}{2}}\right)\right] \tag{2.35}$$

The gradient of $f(\mathbf{x})$ is given by

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \partial_x f(\mathbf{x}) & \partial_y f(\mathbf{x}) \end{bmatrix}^T \tag{2.36}$$

The new target location is estimated iteratively, at iteration $i+1$ it is obtained as

$$\mathbf{x}^{i+1} = \mathbf{x}^i - \eta^i \nabla f(\mathbf{x}^i) \tag{2.37}$$

which is the SPD space steepest descent algorithm. The value of η^i controls the step of the descent (learning rate), it can follow an annealing schedule expressed as $\eta^i = \eta^0(1 - i/N)$, where N is determined empirically. Iterations will continue until convergence, which occurs when $\|\eta^i \nabla f(\mathbf{x}^i)\| < t_{conv}$. For the sake of brevity, the complete development of the SPD gradient equations was omitted here. It is provided in **Section B.2** for the interested reader.

2.3 Conclusions

Here we conclude our panoramic exposition of the *state-of-the-art* methods for detection, matching and tracking. As we have seen, some of the existing machine learning methods for data classification have been successfully applied to the problem of object detection and classification. Dalal and Triggs HOG pedestrian detector or Viola and Jones face detector are important contributions to the field of computer vision that have influenced many other applications. Object detection by classification is very important in the context of surveillance applications, because contrary to the background subtraction methods presented in **Section 1.1.5.1** they allow us to detect objects based on their appearance and not on their motion, this is particularly useful for non-static camera configurations.

The object matching techniques presented in **Section 2.2** are just some examples of the enormous variety of algorithms used for finding correspondences between particular object instances based on their appearance. In the rest of this thesis we will constantly refer to these techniques, specially to the *FoT* algorithm (median flow) and to the tracking methods based on the covariance descriptor.

In the following chapter, we will explore how these methods can be merged to form a cooperation method that adapts to the context and avoids unnecessary calculations while being robust and capable of facing the challenges posed by the long-term tracking problem: occlusions, disappearances, crossings, noise, etc.

CHAPTER 3

Context adaptability and method switching

Contents

Introduction

This chapter proposes a series of context adaptation capabilities to the previously mentioned tracking algorithms. **Section 3.1** initiates with a mechanism proposed in this thesis to evaluate the relevance of color in a local neighborhood and master the difficulties posed by brightness and color saturation changes.

Sections 3.2 and **3.3** introduce some higher-level algorithms that result from *combining* some of the existing state-of-the-art base methods mentioned in the previous chapter. The objective is to obtain a good balance of robustness and accuracy while keeping the required computing power as low as possible to maintain the real-time execution of the base algorithms. This is possible because the idea of method *combination* not forcibly implies incrementing processing work, the algorithms described in this chapter were designed to analyze the context and automatically select the more appropriate tracking algorithm according to it. None of the base tracking methods discussed this far is universally applicable, each algorithm has its advantages and limitations and the performance they offer is most of the times application dependent and there is no algorithm which can be considered *superior* to all the others. Some relevant aspects and threats that a tracking algorithm needs to consider are: the nature of the target, lighting conditions changes, pose and appearance variations, non-rigid deformations and partial or total occlusions. Unfortunately, in many applications such as video-surveillance, there is

no prior information available about the nature of the target (*e.g.* rigid *vs.* non-rigid) and estimating this information on the fly may pose a very challenging problem, this is specially true when objects are detected using background subtraction algorithms instead of an object classifier. This thesis argues that a *good* tracking method may result from the combination of several others, the resulting algorithm should be able to handle a larger list of difficulties than any of the original algorithms while still being fast enough. An enormous processing effort is needed to meet all these goals as real-time execution is commonly compromised when the number of targets to track grows.

The base algorithms considered for this purpose where: (i) KLT optical flow algorithm (**Section 1.1.5**), (ii) Mean-Shift (*MS*) algorithm (**Section 2.2.3**) and (iii) Covariance tracking (*CT*) (**Section 2.2.4.1**).

Optical flow algorithms are of great interest because of their success in determining the displacement of small-sized patches that act very similar to planar surfaces and where it is unlikely for them to suffer non-rigid deformations. The problem with optical flow is that it fails in the presence of occlusions and large and unpredictable displacements. *SIFT* and *SURF* feature points (and their associated descriptors) studied in **Section 1.1.4** are better for handling large displacements, rotations and scale changes, the problem with *SIFT* and *SURF* is that they require the target to be sufficiently textured in order to have a considerable amount of feature points to match. Kernel-based methods such as Mean-Shift (*MS*) and Covariance Tracking (*CT*) construct a global statistical model of the target based on its color and texture, both methods are very good at handling complicated behaviors and tricky non-rigid deformations but they offer a limited capacity to separate the targets from the background, which is possibly their weakest point.

The *FoT* algorithm performs a robust analysis on the optical flow KLT results to detect outliers and estimate the target translation and scale change. Our discussion initiates with a complete overview of the methods used to detect the optical flow outliers **Section 2.2.2**. After it, a detailed description of the cooperation algorithm between the *FoT* and *CT* (*FoT+CT*) is provided in **Section 3.2** together with some experiments that justify its pertinence.

Finally, in the last section of the chapter a scheme of cooperation between *MS* and *CT* (*MS+CT*) is proposed in **Section 3.3**. This algorithm was developed in collaboration with Florence Laguzet, a PhD student college from the *Laboratoire de Recherche en Informatique* (LRI), it integrates my research work to improve the discriminant power and speed of the covariance descriptor and Florence Laguzet's work to ameliorate to the *MS* algorithm by automatically selecting the most discriminant color-space. It is noteworthy to say that very few details about the *MS* algorithm are provided here, more information about the color-space switching mechanisms can be found in [79] and [80].

3.1 Adaptability to brightness and saturation changes

Two different pixels may have the same luminance but different chromaticities, color information provides a complementary measure to luminance that provides an additional discriminative power. In applications like motion analysis where the displacements of a large amount of points are required, color features can be particularly helpful. Those features result from the application of the gradient of Dizenzo defined for vectorial images with the Harris operator using a a procedure very similar to the one used for gray-scale images [106]. Color offers us the opportunity to use invariants and color constancy techniques that provide robustness against illumination intensity and geometry changes [54, 53]. Unfortunately, those properties are exclusive of well-saturated colors and since color features does not depend on luminance variations they may lead us to lose some useful textural information. One of the greatest disadvantages of color features is that they

tend to be noisy when luminance and/or saturation is low. Under such conditions, it is recommended to favor luminance information over color because it is more reliable and requires less computing power.

Within an image sequence, it is possible to experiment an important decrease of lighting intensity. Even worst, the illumination change is generally not uniform over the whole image. Therefore, it is not straightforward to identify the best feature for tracking, either color or luminance information. Important gains of discriminative performance can be achieved measuring the relevance of color based on the saturation and the intensity. Color invariants can be used in an optical flow *KLT*-like tracking method [90, 133] and in covariance matrix descriptors adapting the information to the relevance of color.

3.1.1 *L1* color invariant

The most classical invariant color-space is (H, S, V) where the hue H gives an interpretation of color which is invariant to shadows and specular reflections [54], V is the luminance and S the saturation. While Hue has interesting invariance properties, its value is not so reliable when color saturation is low. In addition, its calculation is more elaborated than luminance and less easy to accelerate since it is based on trigonometric functions.

A very direct way to separate chrominance and luminance from the original (R, G, B) components is to normalize them, obtaining the components (r, g, b) which depend only on the albedo and therefore have no luminance information. For the case of the red color component this normalization is expressed as

$$r = \frac{R}{R+G+B} = \frac{\mathbf{a}_R}{\mathbf{a}_R + \mathbf{a}_G + \mathbf{a}_B} \tag{3.1}$$

representing the albedo.

Starting from rgb, the scalar value $L1 = max(r, g, b)$ represents the more saturated channel while being easier to manipulate compared to a three-dimensional data. As many color invariants, $L1$ reduces the separability between colors. Indeed, as r, g or b, all gray colors become indistinct $\left(r = g = b = \frac{1}{3}\right)$. In addition, $L1$ can reduce the distinction between two colors with same maximum value. The assumption here is that when just small windows of interest are being considered, the probability that two neighbor pixels have same maximum but different colors is low in most natural-image sequences.

Figure 3-1 shows a few images from the sequence *Cardgame* from the ALOI database[1]. The first row displays the classical *RGB* images. Obviously, such features suffer from the illumination changes. The invariant features $L1$ and H are shown on the second and third rows respectively. As expected, the photometric variations are no more visible in these color-spaces.

However, this robustness is reached at the price of a lower separability between colors, especially when their saturation is low. This is noticeable for instance on the eyes of the character drawn on the box of **Figure 3-1**, which become uniformly gray using $L1$. This is also true with Hue since the title of the game can not be read. It is also obvious that the hue produces more noise than $L1$, especially when saturation is low[2]. In a tracking context, this problem can lead to the detection of outliers points and to matching instabilities. The following section introduces the color relevance function, which is used to determine when the color invariant feature can be used or not.

[1] http://staff.science.uva.nl/~aloi/

[2] Hue is artificially represented in range [0-255] although it is an angular value. Therefore some of the noise is due to that representation, and some of the noise is due to ill definition of Hue.

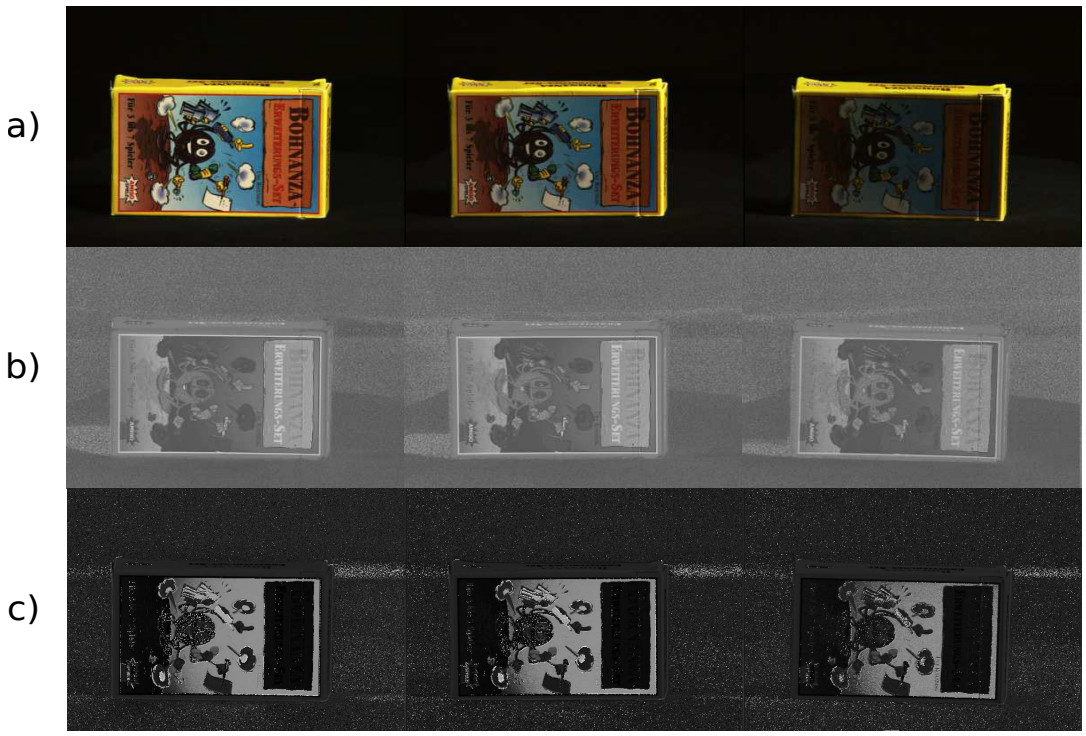


Figure 3-1: a)Original *RGB* images. b) L1 invariant images. c) Hue invariant images.

3.1.2 Relevance of color *vs.* luminance

Carron proposed in [23] a method for color contour detection in the *HSV* space by fusing the information from hue, saturation and value channels while keeping coherency with the introduction of relevance measure which depends on the saturation channel. Hue can be considered as a complement to value and saturation channels and be exploited only when it is considered relevant. Or hue can be privileged over value and saturation and let consider the later only when hue is not relevant. This approach can be extended to other invariants with similar properties.

For each point p there exists a relevance coefficient $\beta\,(p)$ which results from a sigmoid function applied to the saturation S at point p:

$$\beta(S) = \frac{1}{\pi}\left[\frac{\pi}{2} + atan\left(u_\beta\left(S - S_0\right)\right)\right] \tag{3.2}$$

where S_0 is the inflection point and u_β the slope parameter. Under S_0, the luminance is privileged, otherwise the color invariant feature is considered to be sufficiently well-defined and can be used properly.

Coefficient β is defined at each pixel involved in the tracking, in order to use color when it is meaningful and to rely on luminance otherwise.

Color saturation adaptive KLT feature tracking

Tracking is done considering (D_k, I_k) and $(D_{k'}, I_{k'})$, where D is the color invariant for the color-space in consideration (H or $L1$) and I their corresponding luminance images at their respective times k and k'. A physical point P is located at the image in p and p' for frames k and k'.

Each point p and p' has its corresponding coordinates (x, y). For each point p to be tracked, let be a small window of interest $\mathcal{W}$ centred around it, and q a point located in $\mathcal{W}$. The motion undergone by $\mathcal{W}$ is modelled by a function $\delta(p, \mathbf{A})$ where the vector $\mathbf{A}$ describes the deformation of the window from one frame to another. In that manner, the point P at time k' is located at $p' = \delta(p, \mathbf{A})$.

The tracking procedure consists in computing the parameters $\mathbf{A}$ that minimize the following error function

$$\epsilon(\mathbf{A}) = \sum_{q \in \mathcal{W}} \left(\gamma(q, q')\, \epsilon_D(q, \mathbf{A}) + (1 - \gamma(q, q'))\, \epsilon_I(q, \mathbf{A})\right)^2 \tag{3.3}$$

with:

$$\epsilon_D(q, \mathbf{A}) = \|D_k(q) - D_{k'}(\delta(q, \mathbf{A}))\| \tag{3.4}$$

$$\epsilon_I(q, \mathbf{A}) = \|I_k(q) - I_{k'}(\delta(q, \mathbf{A}))\| \tag{3.5}$$

In addition, $\gamma(p, p')$ is the geometric mean of the relevance coefficients for each of the points compared:

$$\gamma(p, p') = \sqrt{\beta_k(p)\, \beta_{k'}(p')} \tag{3.6}$$

After a Taylor expansion for $D_{k'}(\delta(q, \mathbf{A}))$ and $I_{k'}(\delta(q, \mathbf{A}))$ and keeping only the first order coefficients, it yields the following approximation:

$$D_{k'}(\delta(q, \mathbf{A})) = D_{k'}\left(\delta\left(q, \hat{\mathbf{A}}\right)\right) + \mathbf{G}_D\left(\delta\left(q, \hat{\mathbf{A}}\right)\right) J_\delta^{\hat{\mathbf{A}}} \Delta\mathbf{A} \tag{3.7}$$

$$I_{k'}(\delta(q, \mathbf{A})) = I_{k'}\left(\delta\left(q, \hat{\mathbf{A}}\right)\right) + \mathbf{G}_I\left(\delta\left(q, \hat{\mathbf{A}}\right)\right) J_\delta^{\hat{\mathbf{A}}} \Delta\mathbf{A} \tag{3.8}$$

where $\mathbf{G}_D$ and $\mathbf{G}_I$ are the Jacobian matrices of $D_{k'}$ and $I_{k'}$ calculated for both directions x and y. Working with equations (3.7),(3.8) and (3.3), it finally results in the following linearized system

$$\left(\sum_{q \in \mathcal{W}} \mathbf{V}_C \mathbf{V}_C^T\right) \Delta A = \sum_{q \in \mathcal{W}} \gamma \Delta_D^k \mathbf{V}_D + (1 - \gamma)\, \Delta_I^k \mathbf{V}_I$$

with:

$$\Delta_D^k = D_k(q) - D_{k'}\left(\delta\left(q, \hat{\mathbf{A}}\right)\right) \tag{3.9}$$

$$\Delta_I^k = I_k(q) - I_{k'}\left(\delta\left(q, \hat{\mathbf{A}}\right)\right) \tag{3.10}$$

The vectors $\mathbf{V}_D$ and $\mathbf{V}_I$ are defined for an affine motion model as

$$\mathbf{V}_D = \begin{bmatrix} g_x^D & g_y^D & x g_x^D & x g_y^D & y g_x^D & y g_y^D \end{bmatrix}^T \tag{3.11}$$

$$\mathbf{V}_I = \begin{bmatrix} g_x^I & g_y^I & x g_x^I & x g_y^I & y g_x^I & y g_y^I \end{bmatrix}^T \tag{3.12}$$

where g_x^D, g_y^D and g_x^I, g_y^I are respectively the invariant chrominance and the luminance image gradients. Vector $\mathbf{V}_C$ used

for the calculation of the Hessian matrix is defined as

$$\mathbf{V}_C = \left[\begin{array}{cccccc} g_x & g_y & xg_x & xg_y & yg_x & yg_y \end{array} \right]^T$$

where the gradients g_x and g_y are defined as a combination of the chrominance and luminance gradients:

$$g_x = \left[\gamma g_x^D + (1-\gamma)\, g_x^I\right] \tag{3.13}$$

$$g_y = \left[\gamma g_y^D + (1-\gamma)\, g_y^I\right] \tag{3.14}$$

3.1.2.1 Features tracking evaluation

After color conversion when necessary, the feature points are detected by the Harris operator dedicated to color images, then the color channels are loaded in the tracking pyramid. In order to determine which color-space can track more points feature lists were obtained for each of the tested color-spaces and grouped on a single list. Each of these feature lists is built obtaining their gradients in vertical and horizontal directions (applying a Sobel Filter), measuring for each point the Harris operator response, applying a threshold that depends on the global maxima of this function. Finally, local maximum are selected. DiZenzo gradient [106] is employed in RGB color-space as well as Carron2 [23] gradient for HSV.

Sequences are played forward and then in reverse order to verify if points come back to their initial location. Feature positions on the forward trajectory $T_f^k = (\mathbf{x}_t, \mathbf{x}_{t+1}, \cdots, \mathbf{x}_{t+k})$, where k is the number of frames, are compared with their corresponding positions on the backward trajectory $T_b^k = (\hat{\mathbf{x}}_t, \hat{\mathbf{x}}_{t+1}, \cdots, \hat{\mathbf{x}}_{t+k})$. If the calculated distances are less than a defined distance $d_{fb} = 1.5px$ the features are considered as tracked correctly.

For the pyramidal tracking we use the exactly the same parameters (window sizes, number of levels,etc) as the ones described in [19].

Table 3.1: Tracking results

Sequence	Features	I	RGB	(r,g,b)	HSV	H	HP	L1	L1P
Basketball 07-14	2843	1147	1946	1598	1364	667	470	1814	1778
101_l6c 1-3	3179	1642	210	1266	54	696	130	1765	1400
Pedestrian 2 1-10	1061	251	301	386	358	100	336	320	324
Road 16-54	2825	0	37	0	59	0	0	0	14
dtneu_schnee 1-50	2772	2192	2033	63	1521	0	1624	31	2026
Tracking time/feature $[ms]$		0.571	1.331	1.557	1.353	0.517	1.342	0.511	1.307

Five image sequences were considered during experiments, the first two sequences were captured indoors in a controlled illumination environment, 1)**Basketball** [7] has stable illumination conditions, it exhibits colourful and textured matte objects that are somehow easy to follow. 2)**Cardgame** [53] shows a colorful and textured box with a little bit of specular reflections in its borders, it suffers a strong illumination direction change during time. 3)**Pedestrian 2** [73] is a low-resolution outdoors sequence experimenting a relatively strong camera movement, it displays highly and low-saturated

Figure 3-2: Optical flow detected with $L1P$ at *Basketball* sequence second frame. Displacement vectors in yellow are displayed magnified.

objects. 4)**Road**[3] is also an outdoor sequence recorded at very adverse illumination conditions, camera moves in a random fashion harming the stability of tracking, and finally, 5)**Dtneu_schnee**[4] is a traffic scene recorded during a snow fall, coloured objects appear low-saturated, this sequence gives the opportunity to test the tolerance to noise.

The optical flow detected from the first to the second frame of *Basketball* sequence is represented in **Figure 3-2** with magnified yellow vectors, tracking was made with the $L1P$ model. The flow appears quite coherent specially in textured and colored regions. The opposite happens on the upper-left corner, where there is no texture nor colored regions. Those features detected by the $L1$ invariant will be removed by the Forward-Backward error because of their random behavior.

Table 3.1 shows the number of features successfully tracked in each sequence and color-space, the time required to track each feature is also shown. Indeed, the executing times are different from one method to the other because of the color conversion but also on the number of iterations required to converge.

In well-saturated sequences ($Basketball$ and $Pedestrian2$), the luminance I fails to track many of the features that other techniques are able to track employing color information. For example, in the *Cardgame* sequence which suffers an important change of illumination, Luminance is able to track many of the features applying the photometric normalization, while RGB and HSV drastically fail to handle this, while they behave pretty well in many of the sequences.

The relatively high number of features successfully tracked with $L1$ in many sequences prove its tracking capability. As expected, this invariant fails in sequences where color saturation is weak. Tracking with $L1P$ (mixture of $max(r, g, b)$ and I) we are able to handle this situation thanks to the adaptive capability of this technique. $L1P$ maintains the tracking performance of the $L1$ invariant in high-saturated sequences and significantly improves the results for the low-saturated ones. Tracking with (r, g, b) does not improve at all the results obtained with $L1$ while incrementing significantly the calculations.

$L1$ and $L1P$ behave better than their equivalents for the Hue invariant, being able to track more points for all the tested sequences (except the peculiar case of the *Pedestrian 2* sequence), that is probably due to the noisy character of the Hue component, mentioned in **Section 3.1.1**.

A qualitative analysis of the experimental results can be done with the help of color-maps as those built for the *Basketball*

[3]available on `http://vasc.ri.cmu.edu/idb/html/jisct/index.html`
[4]available on `http://i21www.ira.uka.de/image_sequences/#taxi`

sequence and displayed in **Figure 3-4**. Points are assigned a color from a scale that goes from red to green denoting the frame where features were lost. Red for features that were lost on the first frames and green for features correctly tracked during the whole sequence. $L1P$ keeps the good response of $L1$ including areas that neither I nor $L1$ were able to track (like the ear of the man on the right).

Figure 3-5 display the color-maps built for the *Pedestrian 2* sequence, Luminance I succeeds to track the features over the car on the left corner, mostly because this is the most textured region of the sequence. On the other side, it fails to track the people walking at the center of the image that can be regarded as color blobs. Almost any color-space (except Hue) succeeds to track correctly both the car (luminance region) and the people walking at the center.

It is also noticeable in **Figure 3-5** that $L1P$ tracked less color features than $L1$ for this sequence. The influence of the Luminance component, the same that improves the robustness to track low-saturated regions, reduces at some proportion the ability to track color features. It is possible to reduce this phenomenon by finding more appropriate sigmoid function parameters: S_0 and u_β.

Execution times scale-up proportionally to the number of components exploited during tracking. Single component tracking techniques like I, Hue and $L1$ require less time than multiple component techniques. Despite its simplicity in terms of calculations, Luminance I requires almost the same execution time as Hue and $L1$, because of the photometric normalization. Multiple component techniques require more execution time, a price to pay for the robustness and capability to track complementary features.

Figure 3-3 displays the mean forward-backward euclidean distances $d_E\left(x_t, \hat{x}_t\right)$ computed from all the correctly tracked points in each frame of the *Pedestrian 2* sequence. The error increases with the number of frames between x_t and $\hat{x}_t$ in the forward and backward trajectories, but remains lower than one pixel for each method. Note that the use of a single color component ($L1$, H or $L1$) instead of the whole color-space (RGB, HSV or rgb) usually leads to a lower accuracy. Indeed the separability between colors can be lower due to the reduction of components. However, as written in **Table 3.1** the computational costs are reduced with one component and the number of points correctly tracked higher in some difficult sequences, when illuminations changes occur. Mixed approaches $L1P$ and HP provide a better accuracy compared to $L1$ and H respectively while maintaining the invariance properties needed for challenging tracking situations.

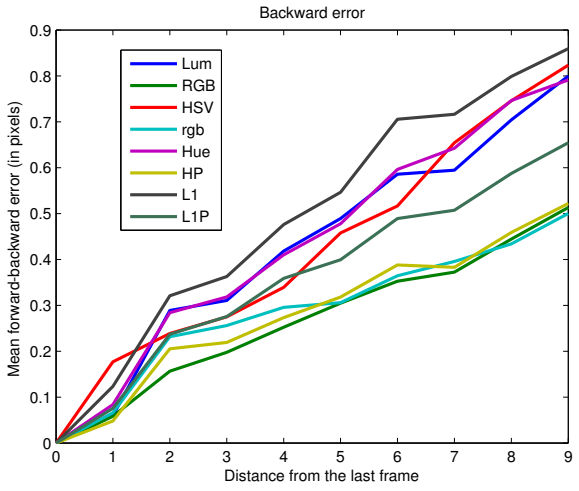


Figure 3-3: Mean forward-backward euclidean distances $d_E\left(x_t, \hat{x}_t\right)$.

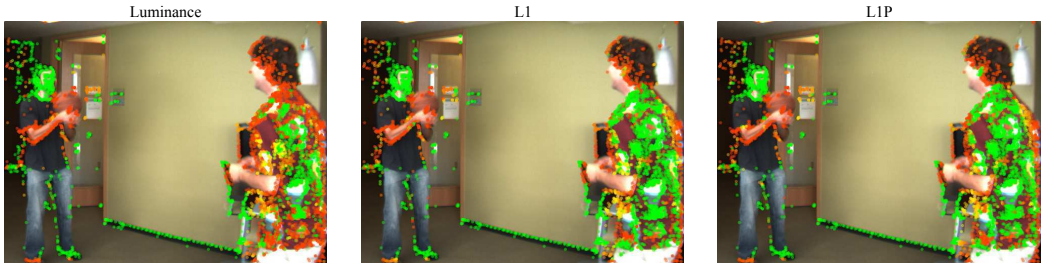


Figure 3-4: Tracking colormaps with Luminance I, $L1$ ($max(r,g,b)$) and $L1P$ (mixed I and $L1$) for the *Basketball* sequence. Points are painted in a color-scale that goes from red to green representing the time the feature was lost

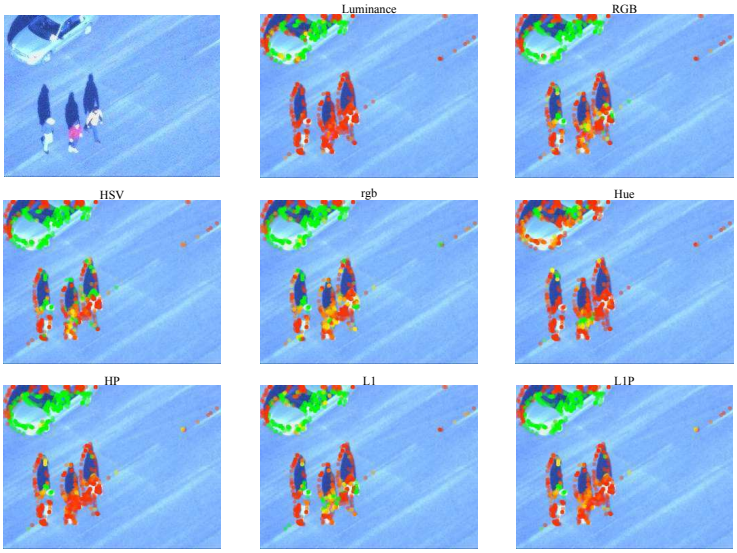


Figure 3-5: Tracking colormaps for the Pedestrian 2 sequence for all the tested color-spaces.

3.2 First cooperation (FoT+CT)

An alternative method to the problem of long-term tracking is proposed here, it is based on the *FoT* algorithm (see **Section 2.2.2**) that analyzes robustly the target's optical flow and the covariance descriptor that models its global appearance (color and texture). As described in **Section 1.1.4**, local feature-based algorithms such SIFT and SURF are very efficient to detect and match a set of key-points no matter their position, scale and rotation but they are somehow limited to inflexible objects. *FoT* robustly estimates the apparent motion of the target and theoritically resisting partial occlusions masking the target for even more than 50%[146] of its surface. The main restriction of the *FoT* method is that the target must be sufficiently unsmooth for the KLT optical flow to be reliable.

Covariance Tracking *CT* (**Section 2.2.4**) appears as an alternative choice that compactly models the target by the correlations within a set of attributes (commonly based on its texture and color). High performances are reached even for low textured objects, since they are represented by a global model. One important weakness of the method is the executing time, this becomes more significant when the algorithm is executed using its exhaustive exploring modality. However, *CT* is accelerated by coupling it with other tracking methods (such as particle filtering [87] for example) that confine the space

of research using the information provided by the target state (*i.e* size, speed, position).

The interest of mobile and real-time video tracking applications is to detect and follow not only a single type of target but a variety of them. The most common types in surveillance applications are: faces, pedestrians, clusters of people and vehicles. In addition, the desired tracking algorithm should be able to deal with very adverse filming conditions: outdoor or indoor, changing angles and scales, with a static or a moving camera, in luminous or dark scenes, etc. In order to have a good adaptivity to the context while still preserving the real-time execution of the algorithm this section presents a new method that combines the complementary advantages of the *FoT* and *CT* tracking algorithms by changing adaptively with the context. This algorithm is ideal for dealing with target occlusions without any presumptions about the nature of the target or their type of motion. This cooperation method is denoted from now on as *FoT+CT*.

3.2.1 *FoT* outliers detection

The detection of outliers is very important to improve the performance of the *FoT* algorithm. Some popular methods for this purpose are:

a) **Forward backward method**: Many feature point tracking algorithms detect failures by describing each tracked point by a surrounding patch $\mathcal{W}$ which is compared from time t to $t+1$ using template matching techniques such as *MSE*, *NCC* and *SSIM*. These error measures allow us to detect failures caused by occlusions or rapid movements because the feature's local patch appearance changes drastically from one frame to the next. But slowly drifting trajectories, where a tracker gets attracted by a nearby location that at this particular instant looks similar to the trackers patch. It is important to emphasize the observation that not all pixels in a local neighborhood move exactly the same. For this problem the affine *KLT* model was introduced (see **Section 1.1.5**) to estimate more complex geometrical transformation parameters than simple translations. It is worth considering that the affine KLT method is constrained to planar targets only.

   The Forward-Backward (*FB*) error measures the coherency of a set of feature point trajectories. The sequence $S = (I_t, I_{t+1}, \cdots, I_{t+k})$ is an image sequence and $\mathbf{x}_t$ expresses the location of a point at time t. With a feature point tracker such as *KLT* the point $\mathbf{x}_t$ is tracked forward for k frames defining the trajectory $T_f^k = (\mathbf{x}_t, \mathbf{x}_{t+1}, \cdots, \mathbf{x}_{t+k})$, where f indicates the forward direction and k expresses the length. To estimate the reliability on the trajectory T_f^k, a validation trajectory is constructed, the point $\mathbf{x}_{t+k}$ is tracked backward up to the first frame producing the trajectory $T_b^k = (\hat{\mathbf{x}}_t, \hat{\mathbf{x}}_{t+1}, \cdots, \hat{\mathbf{x}}_{t+k})$. The Euclidean distance between this two trajectories expresses the *FB* error as

$$FB(T_f^k, T_b^k) = \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|. \tag{3.15}$$

   Under challenging conditions such as when the motion of the target is faster than what the tracking algorithm can handle, the displacement depends on factors not considered by the tracking model and becomes to some extent random. *FB* error works pretty well for such conditions, because it is very unlikely that the tracker will follow the same path in the opposite sense.

b) **Neighborhood consistency verification**: Neighboring feature points are expected to move in a similar way, but as mentioned for the forward-backward case when trackers fail random displacements that have no relation with their

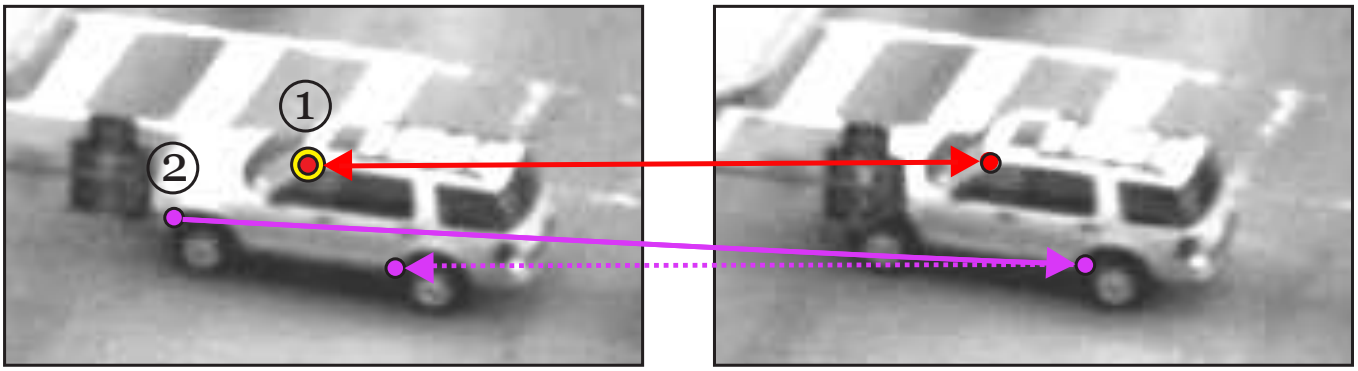


Figure 3-6: Inconsistent trajectories are penalized by the Forward-Backward error. Points that are visible throughout the sequence are consistently tracked forward and backward in time (point 1), while points that are occluded have inconsistent trajectories (point 2).

neighbors appear more often. Wendel *et al*. presented in [146] their *Nh* predictor which is implemented as follows. For each feature point i, a neighborhood consistency score S_i^{Nh} is obtained, based on the behavior of the neighborhood N_i which contains the four neighbors of i (edges and corners in the bounding box have only three or two neighbors). S_i^{Nh} accounts for the number of local trackers that have similar displacements in comparison to i, this value is computed as,

$$S_i^{Nh} = \sum_{j \in N_i} \left[\|\Delta_j - \Delta_i\|^2 < \varepsilon_{Nh} \right], \tag{3.16}$$

where Δ_i and Δ_j are the displacements of the local tracker i and its j-neighbor, while ε_{Nh} represents a threshold of the displacement difference. A local tracker is labeled as consistent when $S_i^{Nh} \geq \theta$, the suggested value of θ is 1, and the displacement threshold is $\varepsilon_{Nh} = 0.5$ pixels.

c) **Markov predictor**: The Markov predictor (*Mp*) is used to model the behavior of a local tracker in time. A two-state Markov chain (inlier and outlier) is used to predict the status of a local tracker depending on its state in the previous time instance and a a set of transition probabilities which are calculated incrementally from frame to frame. Each tracker i has its own transition probabilities which are included in the transition matrix $\mathbf{T}_t^i$ structured as

$$\mathbf{T}_t^i = \begin{bmatrix} p^i(s_{t+1}=1|s_t=1) & p^i(s_{t+1}=1|s_t=0) \\ p^i(s_{t+1}=0|s_t=1) & p^i(s_{t+1}=0|s_t=0) \end{bmatrix}. \tag{3.17}$$

In equation (3.17), s_t represents the current state of the local tracker (inlier=1, outlier=0). Each of the transition probabilities are complementary, and the terms contained in both columns must be equal to 1. The state at $t+1$ depends on the probability of the state at the current time t and the transition matrix. This is calculated as product

$$\begin{bmatrix} p^i(s_{t+1}=1) \\ p^i(s_{t+1}=0) \end{bmatrix} = \mathbf{T}_t^i \times \begin{bmatrix} p^i(s_t=1) \\ p^i(s_t=0) \end{bmatrix}, \tag{3.18}$$

where the left hand of the equation expresses the probabilities of the trackers next state, considering that at the current time the probabilities are $p(s_t=1)=1$ and $p(s_t=0)=0$ if the current state is inlier (likewise for the opposite case).

The transition probabilities in $\mathbf{T}_t^i$ are updated as follows

$$\begin{aligned} p^i &= (s_{t+1} = 1 | s_t = 1) = \frac{n_{11}^i}{n_1^i} \\ p^i &= (s_{t+1} = 1 | s_t = 0) = \frac{n_{01}^i}{n_0^i} \end{aligned} , \tag{3.19}$$

where n_1^i and n_0^i are the relative frequency for the local tracker i is an inlier or an outlier. In turn, n_{11}^i and n_{01}^i reflect the relative frequencies of transitions events on tracker i (e.g., n_{01}^i represents the relative frequency of the tracker i passing from the outlier state to being considered an inlier). The current state of the tracker is verified measuring the error between the trackers displacement and the global target motion and applying a threshold.

3.2.2 *FoT+CT* algorithm description

FoT+CT is summarized in **Algorithm. 10**, it starts by analyzing the sequence of images I_t (t stands for the time-stamp), targets are appended to an initially empty tracking list denoted as `objList`. Objects are selected manually or detected by a background subtraction method such as Sigma-Delta (see **Section 1.1.5.1**) when the camera is static. In moving camera sequences, targets are detected by means of an object classifier (see **Section 2.1**). Other methods are conceivable such as the disparity map when stereo-vision is available.

In the *FoT+CT* cooperation method an object detection algorithm is executed each t_{detect} frames (procedure `objDetection` line 4), unless they are provided by an specific object detector, the nature of the targets provided by background subtraction algorithms is generally unknown. Initially, the detected objects are considered to have a rigid motion and to contain key-points, therefore the tracking is initialized by *FoT* (see **Section 2.2.2**). Features points extracted in each rigid object are pushed into a list (line 6), and, each of the n features inside this list is tracked with the KLT pyramidal method (line 6) to compute its motion $\mathbf{X}_k$. Each target the global residual flow ($\mathbf{X}_R$) is then calculated, the amount of flow which differs from the median flow ($\mathbf{X}_{median}$) is

$$\mathbf{X}_R = \frac{\sum_{k=1}^{n} \mathbf{X}_{median} - \mathbf{X}_k}{n}, \tag{3.20}$$

which represents a measure of the coherence of the optical flow.

A low residual flow indicates that the motion of the points is consistent, *i.e.*, the trackers have correctly converged and agree for a similar motion. In that case, the *FoT* is well adapted to the object, and the latter is confirmed as rigid ($\texttt{obj}_{rigid}$ in **Algorithm. 10**). At the same time, the current covariance matrix $\mathbf{C}_t$ and its dissimilarity $\mathcal{D}_t$ with the previous model $\mathbf{C}_{t-1}$ are computed, for three reasons:

- preparing a possible switch to *CT* in case of failure of the *FoT*,
- confirming the correctness of the tracking or,
- preventing from a drift of the target during the tracking.

While an object remains classified as rigid, the *FoT* is run.

Abrupt increments of $\mathbf{X}_R$ can appear for several reasons. The most popular ones are occlusions or object's appearance changes, illumination changes such as shadows or highlights created by luminous sources. Under such situations, the *FoT* may fail because of the ill-conditioning of the matrices involved in the KLT tracking procedure.

Algorithm 10: Tracking methods cooperation

Input: Image Sequence $I_t \; \forall t$

1 `objList` $\leftarrow$ `objDetection`(I_0)
2 **for** $t > 0$ **do**
3 *// New detections and object verifying each t_{detect} frames:*
if t `multiple of` t_{detect} **then**
4 `objList` $\leftarrow$ `objDetection`(I_t)
5 *// Track each object features by KLT:*
foreach `obj` $\in$ `objList` **do**
6 `pyramidalKLT`$(I_x, I_y, I_{t-1}, I_t,$ `featuresList`$)$
7 *// Analyze the optical flow:*
foreach `obj` $\in$ `objList` **do**
8 **if** `obj`$_{rigid}$ = *TRUE* **then**
9 `FoT(obj)`
10 *// Classify Object as Rigid/Non-rigid:*
`obj`$_{resFlow} \leftarrow$ `residualFlow(obj)`
11 *// Calculate CT model dissimilarity:*
$\mathbf{C}_t \leftarrow$ `getCovarianceMatrix(obj,`I_t`,`$\mathcal{V}$`)`
$\mathcal{D}_t \leftarrow$ `covDissimilarity`$(\mathbf{C}_t, \mathbf{C}_{t-1})$
12 **if** `obj`$_{rigid}$ = *FALSE* ***OR*** $\mathcal{D}_t > \mathcal{D}_{max}$ ***OR*** `residualFlow(obj)`$> \mathcal{R}_{max}$ **then**
13 *// Run Covariance Tracking:*
`obj`$\leftarrow$ `covTracking`$(obj, I_t, \mathcal{V})$
14 *// Update model $\mathbf{C}_{t-1}$ for the next frame:*
$\mathbf{C}_{t-1} \leftarrow$ `covUpdateModel(obj)`

When a tracker starts losing a target, an increase of the $\mathbf{X}_R$ is generally observed. The quality of the tracker is monitored as well by constantly comparing the appearance of the target in the current location against the model. If they are unalike (the covariance dissimilarity $\mathcal{D}_t$ is higher than the threshold $\mathcal{D}_{max}$), the algorithm activates the covariance exhaustive matching instead of *FoT*. An object can also be classified as *non-rigid* when its residual flow $\mathbf{X}_R$ reaches a limit $\mathcal{R}_{max}$, this means that the *CT* algorithm should be a better option to track this specific target. This state is verified frequently by testing the performance given by *FoT*. Indeed, when both *FoT* and *CT* behave correctly, (i.e., the residuals are low and the covariance dissimilarity is low), then *FoT* is logically preferred because it runs faster.

Finally, whatever the current tracking method is (*FoT* or *CT*), the covariance model $\mathbf{C}_{t-1}$ is updated considering the appearance at the newest position (line 14), in order to prepare the next frame-to-frame matching.

Additionally to the algorithmic description of **Algorithm 10**, a description of the switching mechanism for a single target is provided graphically by the flow-chart in **Figure 3-7**.

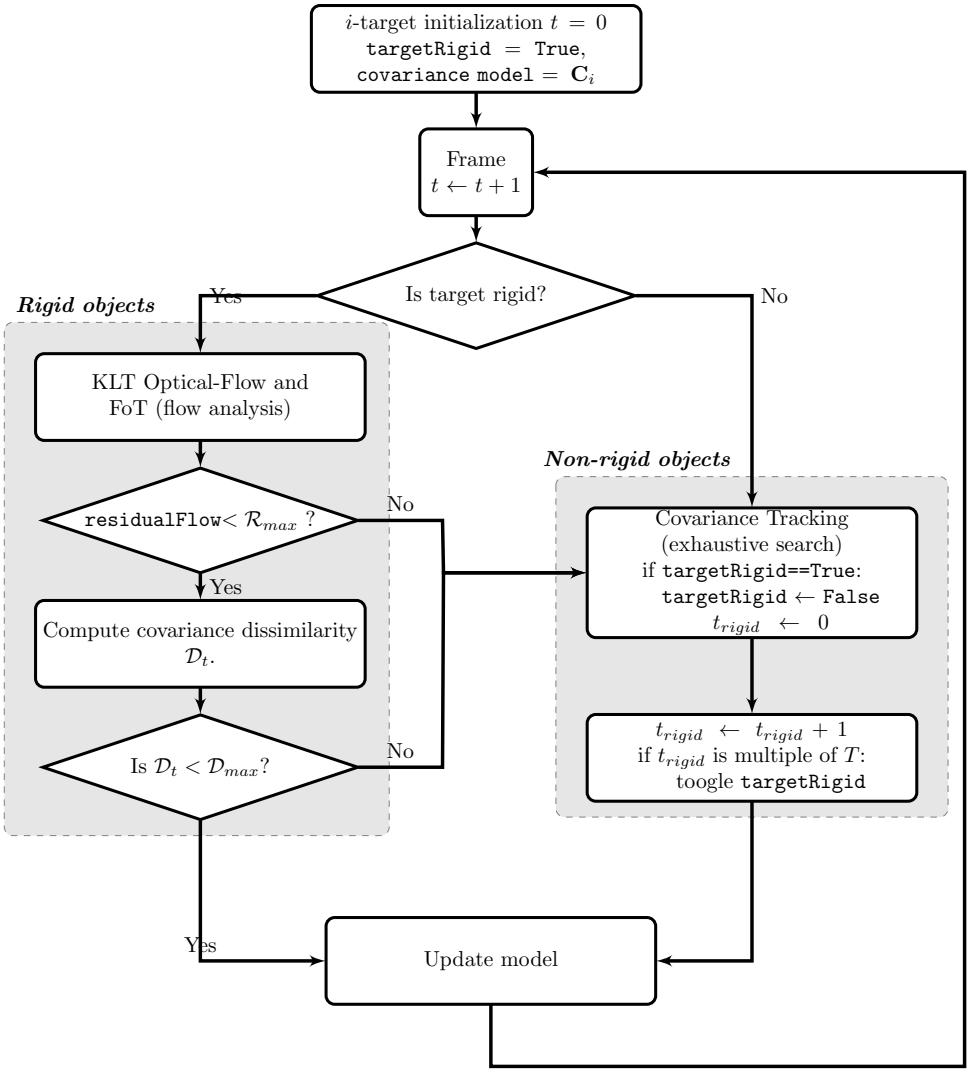


Figure 3-7: Flow chart of the *FoT+CT* algorithm.

3.2.3 *FoT+CT* evaluation experiments

To evaluate the performance of the *FoT+CT* cooperation algorithm a set of very challenging sequences was selected. Most of them include partial and complete occlusions, targets moving very fast or very adverse imaging conditions where the objects appear ill-defined and/or blurred. **Table 3.2** summarizes the set of sequences included in this dataset. The first three columns denote the sequence name, the total number of frames considered for the tests and their citation or URL for download. The three sub-columns on the table (under *Last Frame* sub-heading) indicate the overall results obtained by each method: *FoT*, *CT* and *FoT+CT* represented by the frame number at which each method stopped. The last group of sub-columns (under the *FoT+CT* heading) writes down the behavior of the switching mechanism: the percentage of frames the algorithm spends using the *CT* exhaustive search algorithm (%CT), the average number of milliseconds demanded by each method (*FoT* and *CT*). The rightest column indicates the total speed-up obtained by using the switching mechanism in comparison to executing *CT* alone for the entire sequence.

Sequence	# frames	Dataset	Last Frame			*FoT+CT*			
			FoT	CT	FoT+CT	% CT	FoT[ms]	CT[ms]	Speed-up
Sylvester	1344	[118]	620	1344	1344	40%	2.8	4.8	25%
Auto	945	[73]	220	945	945	34%	2.9	4.8	26.5%
Motocross	2665	[73]	65	958	958	36%	4	6.3	24%
Panda	3000	[73]	95	1015	1015	40%	2.2	3.7	24.3%
Pedestrian1	140	[73]	95	140	140	71%	2.7	4.8	12.7%
PETS 2001 Dataset 1	338	PETS 2001[5]	42	338	338	60%	2.9	4.5	14%

Table 3.2: Tracking results. Last frame reached by each method is indicated. Detailed results are also represented for the case of *FoT+CT* method: percentage of frames *CT* was preferred over *FoT*, the average time each method demanded and the speed-up gained using *FoT+CT*.

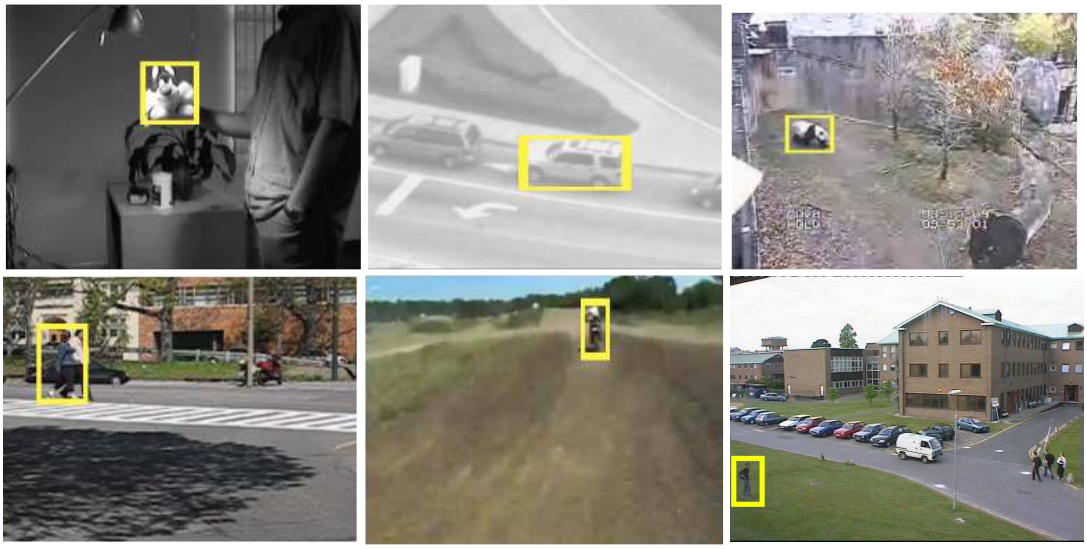


Figure 3-8: Targets location at first frame for each sequence.

A list of aspects constitute the evaluation criteria: the length of the sub-sequence where the tracker works successfully , the tracking precision, the stability and the speed.

FoT alone is a bad long-term tracker because it does not incorporates a drifting recovery mechanism. This is observable on columns five to seven from **Table 3.2**. By monitoring the coherence of the flow $\mathbf{X}_r$ (in equation 3.20) the appearance dissimilarity (the distance to the covariance model $\mathcal{D}_t$) the *FoT+CT* algorithm is able to inform the tracker when an occlusion or a target drifting is occurring, activating immediately the *CT* exhaustive search algorithm which should be able to re-detect it if the target is still visible or recollect it when the temporal occlusion ends.

Very frequently, *CT* alone is enough to follow the target from end to end, however, the trajectories provided by this method tend to be very shaky and in some cases they conduce to a drift of the tracker when it gets distracted by other objects on the background. *FoT* oscillates less, but is not equipped to deal with total occlusions. *FoT+CT* in turn inherits the stability of *FoT* and the robustness of *CT* and describes smoother trajectories that tend to drift less, not only *FoT+CT* accomplishes better the task, most of the time it turns faster than executing *CT* alone.

Figure 3-8 shows the first frame and the selected target for each of the six sequences tested (see **Table 3.2**). We provide here a careful description of sequence behavior for each of the three methods evaluated (*FoT*, *CT* and *FoT+CT*).

- **Sylvester**: This sequence displays a plush toy handled by a man, all images were recorded in gray-scale. The target is is easily recognizable because it contains highly contrasted black and white patches. The evolution of the target appearance is shown in **Figure 3-9a** which contains some snapshots taken using the *FoT+CT* tracking algorithm. It is easily noticeable that the target behaves pretty much as a rigid object throughout the sequence and that the only difficulty posed to the tracker is to follow the capricious changes of the target's position, scale and perspective. The frames where the tracker is not adequately centered are shaded in red, thanks to the *CT* algorithms the tracker is able to recover it.

- **Auto**: Filmed in gray-scale, this sequence displays a vehicle advancing on a road and being followed by a moving camera on the air. So, it is a rigid target that preserves its appearance unchanged from the beginning to the end of the sequence. Even though, this sequence is way more complicated than the previous one, the reason is that the camera moves haphazardly and that the target suffers partial and total occlusions on multiple occasions. Moreover, there are very similar vehicles that appear from time to time near to the target. The evolution of the target captured by the *FoT+CT* tracker is shown in **Figure 3-9b**.

- **Panda**: This video shows a panda walking in its zoo confinement. The video was filmed in color by a still camera that pans and zooms in and out at some points in time. In the images, there are some zones textured pretty much alike to the panda. These zones can cause strong confusions when the tracker looks for the target exhaustively. For example, the text on the bottom is a potential distraction if the target moves around it. Most of the time, *FoT* is executed except for the moments when the panda turns around or when it passes behind the tree on the top right corner. The use of the *CT* algorithms becomes indispensable, allowing the algorithm to re-acquire the target once it has been lost. The evolution of the target appearance and the tracker behavior is illustrated in **Figure 3-11a**.

- **Pedestrian 1**: This sequence is probably one of the most demanding one of the testing dataset. And it is here where the *FoT+CT* algorithm probes its utility more clearly. Both *FoT* and *CT* are clearly unable to follow consistently the target when executed individually. The real difficulty strives in the unpredictable movements of the camera which cause *FoT* to lose the target almost immediately. On the other hand, *CT* is capable of recovering in all the occasions it gets lost, but due to the so spoken unsteadiness of the *CT* algorithm which is even more severe due to the camera movements the results are not very convincing. The stability of the *FoT+CT* is remarkable, even the violent shaking of the camera, the tracker is able to retrieve a valid target position for most of the frames correcting immediately all the small drifts of the tracker. **Figure 3-11b** shows the sub-images captured by the trackers bounding box at each frame of the sequence.

- **Motocross**: This sequence appears to be a recording from a motocross competition. The camera is probably mounted on a bike that is always behind the target. The recording results are very blurred and shaky as the camera advances and jumps constantly. Matching objects under such conditions is a an arduous task. Similar to the previous sequence *FoT* fails immediately, and while *CT* is able to retrieve the target it usually takes some frames to glance over the complete set of possible locations and bump into to a promising one. For some frames the optical flow is enough to estimate the displacement of the target, this stabilizes the tracker and reduces the computing cost. When the target is lost because it disappears from the camera field of view or because it moves violently, the *CT* is acts in response

to retrieve it. Most of the frames in this sequence operate in this modality. **Figure 3-10** shows the evolution of the target along the sequence, the difficulty posed by this sequence is easily observable.

- **PETS2001 Dataset 1**: This sequence was captured by a fixed surveillance camera and despite its appearing simplicity some interesting tracking difficulties arise. The first one is attributed to the low-resolution of the target, the other intriguing condition takes place when target traverses from the grass to the pavement, causing the background color to change drastically from green to gray. As the covariance model assigns the same weight to every pixel inside the targets bounding box, this forcibly has an impact on the covariance descriptor. *FoT+CT* is able to compensate these tracking problems and retrieves the target even when it passes behind the lighting post using the *CT* algorithm, and thanks to the optical flow analysis of *FoT* it is able to estimate the displacement of the target even if the color of the background changes. The described phenomenons are noticeable in **Figure 3-11c**.

(a)

(b)

Figure 3-9: *Sylvester* and *Auto* sequences.

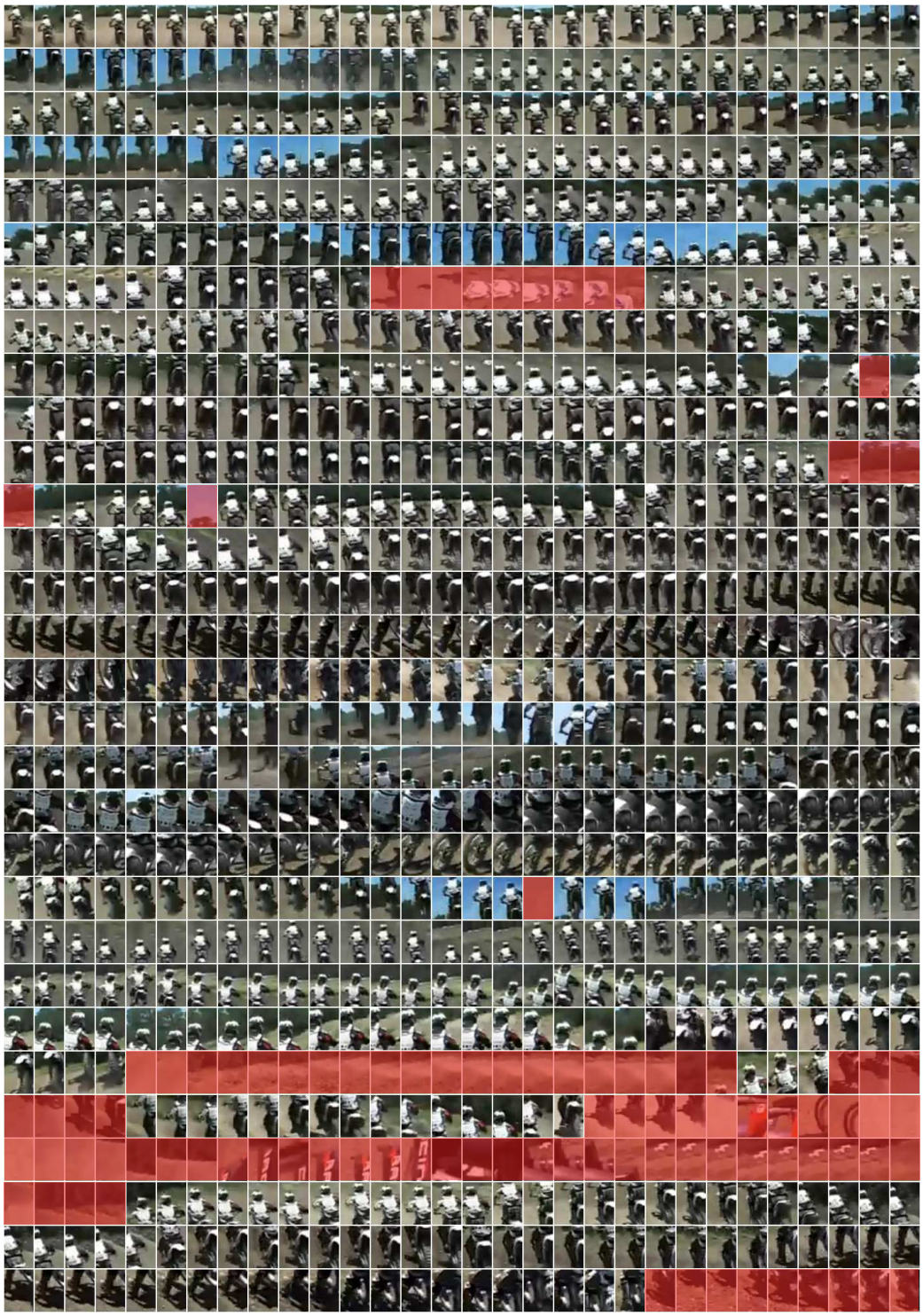
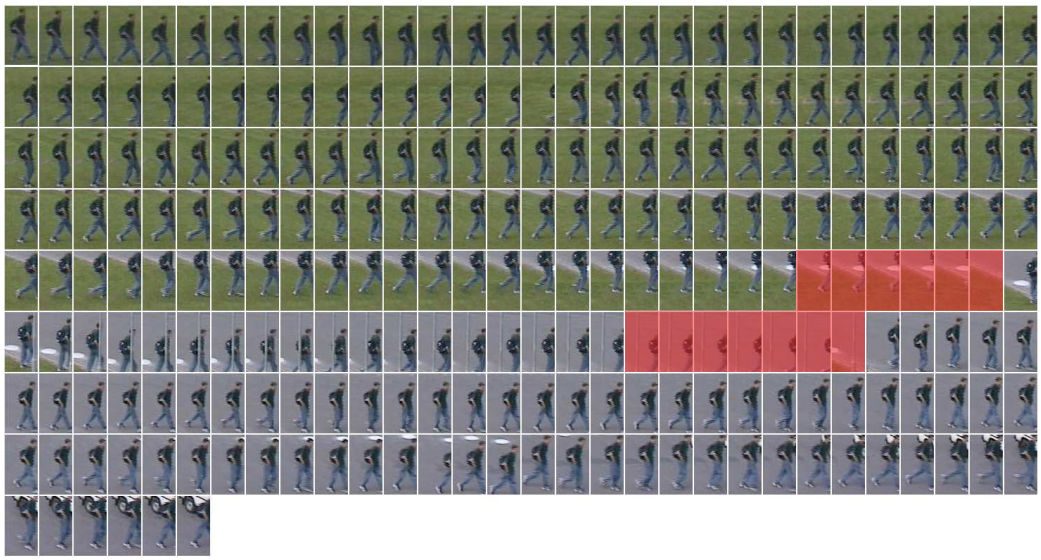


Figure 3-10: *Motocross* sequence

(a)

(b)



(c)

Figure 3-11: *Panda*, *Pedestrian1* and *PETS 2001 Dataset 1 sequences*.

3.3 Second cooperation (MS+CT)

As for the case of the previous section *MS* and *CT* have complementary advantages too. *MS* represents the target by its color distribution, a model which is almost insensitive to geometric distortions and valuable property when tracking non-rigid targets. *MS* runs fast but similarly to *FoT* it is unable to handle large motions and occlusions. *CT* enjoys a greater separability power and is applicable with many kinds of objects. Contrary to *FoT* and *MS*, *CT* is comfortable with large displacements, but it usually turns out to be less time effective than *MS* and other tracking methods. In order to reduce the computation times of *CT* without a robustness loss, this section details a cooperation algorithm between *MS* and *CT* (denoted as *MS*+*CT*), where the covariance dissimilarity is used as an indication to check the accuracy of *MS*, then and where the *CT* algorithm is run to retrieve the target after it is lost or it exits from an occlusion.

3.3.1 *MS+CT* algorithm description

Kernel-based methods of the type of Mean-shift [30] are usually well appropriate to track objects with non-rigid motion and low textural or structural contents, because they rely on a overall statistical distribution that is invariant to some appearance deformations. The price to pay is a decrease of the robustness. Several attempts have improved the method by background subtraction, color space switch [78] or by using a spatio-colorimetric histogram [56]. *MS* follows a minimization procedure which assumes small inter-frame motions, when *MS* fails for some reason (like occlusions and false convergence of the *MS* minimization scheme) *CT* is asked to retrieve the target. To be able to do so, the exhaustive search *CT* method (described in **Section 2.2.4.2**) is selected. Once the target has been retrieved and that the tracking conditions are back to normal, it is possible to give back the control to *MS* and carry on.

This cooperation scheme is deterministic and no assumption is made on the motion model of the target. Failures of *MS* and *CT* can be detected but no explanation about their origin is required as the retrieving mechanism does not depend on it.

Figure 3-12 represents the flow chart of the *MS*+*CT* cooperation algorithm. This figure can be easily decomposed in the following steps:

1. **Initialization**:. At time $t = 0$ a detection algorithm provides the windows of interest where it is assumed the targets are located, here is where the *MS* is started, at this point the covariance descriptor of the target $\mathbf{C}_t$ is computed too.

2. **Standard tracking**:. Since *MS* is the less expensive algorithm it is preferred over *CT* when possible. However, the quality of the *MS* tracking algorithm is verified constantly during the sequence. The appearance of the target j at its current location $\mathbf{x}_j^t$ at time t is represented by the covariance matrix $\mathbf{C}_j^t$. This descriptor is compared with the target model $\mathbf{C}_j$ using the Riemannian metric of equation (2.30) to confirm the correctness of the tracking and to prevent draftings of the tracker. If $\mathcal{D}_t$ represents the dissimilarity of the current target location appearance with respect with respect to the target model, when $\mathcal{D}_t$ is low the location of the target is confirmed and the tracking continues with the same method. The covariance model of the target $\mathbf{C}_j$ is updated constantly preparing it to the next frame-to-frame matching. It is very important to highlight that the covariance matching is not run in all frames, *i.e.* there is no exhaustive search comparing multiple candidate locations. At this point, the target location has already been found

by *MS*, allt that the algorithm needs to do is to compute the covariance descriptor $\mathbf{C}_j^t$, compare it against the target model $\mathbf{C}_j$ and update it. All these tasks demand little extra work.

3. **MS failure**:. Abnormal tracking conditions such as occlusions, illumination changes, large motion or appearance variations. Lighting changes may alter drastically the histogram of the target if the employed *MS* color-space is not invariant to those changes. When the tracker begins to loose the target, the covariance dissimilarity $\mathcal{D}_t$ becomes higher than a threshold $\mathcal{D}_{max}$, and the covariance matching (*CT*) is run in order to re-acquire the target. When the new $\mathcal{D}_t$ is back to a tolerable low-level *i.e.* the target is considered re-acquired and controls returns to *MS* which is less time-consuming. Otherwise, *CT* executes as long as $\mathcal{D}_t$ remains higher than $\mathcal{D}_{max}$. If *CT* is unable to retrieve the target and $\mathcal{D}_t$ remains high after a long time T, the target is assumed to be definitely lost.

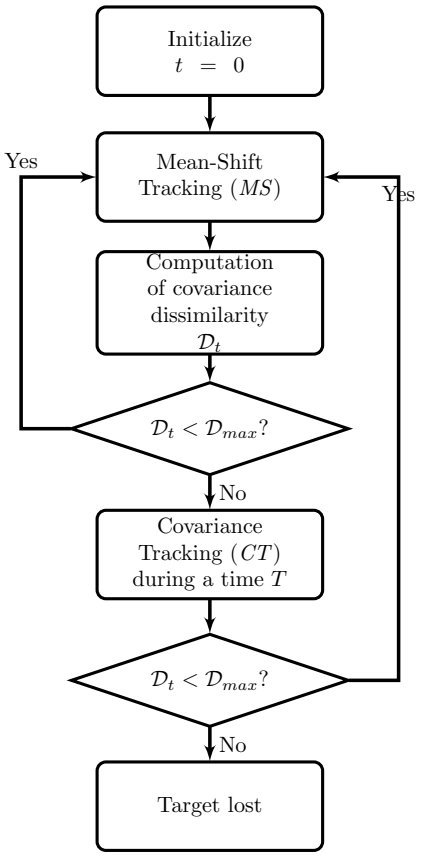


Figure 3-12: Flow chart *MS*+*CT*

3.3.2 *MS+CT* evaluation experiments

In this subsection the performance of the **MS+CT** cooperation algorithm is evaluated for a number of tracking sequences.

- **Pedxing**: In this sequence, the *CT* (shown in red) fails at $t = 7574$, while *MS+CT* correctly tracks the pedestrian during the whole sequence (see $t = 7672$). When the target is not occluded, *MS* is generally more precise than *CT*, when the pedestrian passes behind the panel at $t = 7574$, it is correctly handled by switching from *MS* to *CT*. For the less-precise feature vector configurations, *CT* alone is distracted by another pedestrian before the occlusion actually happens.

- **Panda**: At the beginning of the sequence (from $t = 1$ to $t = 80$) *MS* behaves correctly. But suddenly the panda turns on itself and its appearance changes drastically, this is where the value of $\mathcal{D}_t$ increases and becomes higher than $\mathcal{D}_{max}$. This value remains high during several successive frames (see $t = 190$), when the back of the panda is viewed by the camera. *CT* is charged of tracking during all this period of time, while $\mathcal{D}_t$ remains high. When the panda turns of itself once again, its appearance is closer to its initial appearance, $\mathcal{D}_t$ decreases and control returns to *MS* ($t = 300$ and $t = 400$). This type of events occur several times throughout the sequence ($t = 800$ and $t = 900$). The cooperation of both algorithms succeeds in two aspects: it choses the fastest and more precise algorithm when it is reliable (*MS*) and rapidly switches to the algorithm that is better adapted to deal with its difficulties (*CT*).

- **Motocross**: Due to the large motion of the target, the *MS* fails in frame $t = 150$ and never retrieves the target. In the cooperation procedure, the dissimilarity $\mathcal{D}_t$ allows to identify this problem and *CT* is preferred over *MS* most of the time. When the target is retrieved, *MS* is able to track the target for some time. Running *CT* alone is generally less precise in the fragments of the sequence when the target moves smoothly. For some feature configurations *CT* alone can even fail.

- **Carchase**: In this sequence *MS* loses the target most of the time. For example, at $t = 365$, lighting conditions change and *MS* minimization procedure fails. At $t = 390$ the target is occluded. On the other hand, *CT* is sometimes distracted by other cars (as shown in frames $t = 50, 413, 1250$ and $t = 2590$). With the cooperation of both algorithms several difficulties are handled better: the total occlusion by a bridge between $t = 365$ and $t = 390$, the partial occlusion and blur at $t = 413$, scale change at $t = 2430$, etc. The association of *MS* and *CT* provides more precise results than *CT* alone for standard situations (*i.e.* free from occlusions) obtaining an overall more robust algorithm than *MS* when the target is lost due to occlusions or lighting changes.

Table 3.3: Analysis of the *MS*+*CT* cooperation procedure. Results were obtained on a *Nehalem* processor. The table shows for each sequence: the target size at the initialization point; the percentage of frames (%fr) for which the *MS* and *CT* algorithms are run; the total number of cycles per pixel (*cpp*) spent by each section of the algorithm (*MS*, *CT*, Covariance model Update, computation of the similarity) during the whole sequence.

Sequence	Target	# frames	*MS*		*CT*		Model Update	Similarity	Time
	size		*cpp* ($\times 10^6$)	%fr	*cpp* ($\times 10^6$)	%fr	*cpp* ($\times 10^6$)	(*cpp* $\times 10^6$)	[ms]/fr
Pedxing	93×35	189	7.20	70	60.59	30	1.07	2.45	10.1
Panda	23×28	940	2.48	88	8.74	12	0.57	0.33	1.6
Motocross	64×47	2665	5.07	12	30.04	88	0.005	1.06	10.6
Carchase	45×97	2999	0.99	13	19.19	87	1.03	1.44	7.3

Figure 3-13(a)-(d) shows the initial and the final target position for the set of tested sequences. *MS* alone is marked in pink, *CT* in red and the *MS*+*CT* method in blue or green depending on which method has been used on that frame. **Figures 3-14**, **3-15** and **3-16** are helpful to see where the *MS* (in pink) fails and where each one of the base methods *MS* (blue) or *CT* (green) are being used.

Table 3.3 analyses for each sequence the number of cycles per second (cpp) spent by each section of the *MS*+*CT* switching algorithm throughout the whole sequence. The algorithm can be decomposed in several sections, *MS* and *CT* (which are never executed simultaneously), the model update and the similarity criterion which execute at every frame. In addition, the percentage of frames for which *MS* and *CT* are selected with respect to the sequence length is shown. In sequences where the camera is more stable, such as *Pedxing* and *Panda*, *MS* is executed most of the time as long as motion is regular and free from occlusions. If those conditions are not met, *CT* runs and retrieves the target location. When the camera is moving (as it is the case of *Motocross* and *Carchase* sequences), the motion of the target is tricky and unpredictable. Therefore, *CT* is run more often.

The color tracking by contextual switching has two main advantages:

1. **Robustness**: *CT* helps the *MS* tracking algorithm when it is deficient due to occlusion, tricky motions and appearance changes.

2. **Speed**: *MS* is faster the computation time is generally reduced compared to *CT* alone. The more often *MS* is run the faster the tracking. Consequently, the cooperation procedure is globally less time-consuming when the camera is still and when the target is not occluded.

3.4 Conclusions

The first group of contributions of this thesis to the *state-of-the-art* has been discussed. The *L1* color invariant is our first proposal to improve the quality of the optical flow. The second contribution presented here consist in modifying the equations used by the KLT optical flow algorithm to handle brightness and color saturation changes better. The approach presented here measures the relevance of color using a sigmoid function that depends on the saturation of the pair of pixels being compared. The experiments associated to these contribution demonstrate its utility to increment the number of reliable features to track at the same time that the magnitude of the tracking errors is reduced.

The second set of contributions treated in this chapter is the pair of *combination* methods *FoT*+*CT* and *MS*+*CT*. All the three base algorithms: flock of trackers (*FoT*), mean-shift (*MS*) and covariance tracking (*CT*) are known to be capable to handle non-rigid targets, but each one obeys its own limitations: *FoT* is more suited to objects which have enough salient feature points and which undergo rigid motion, *MS* is preferably used when the color histogram is discriminant enough, and it doest not require any salient feature. In terms of computation time, *FoT* and *MS* are more efficient than *CT*. In regular situation the *FoT* algorithm is reliable and describes smooth trajectories for smoothly moving objects but fails when the target is occluded. In turn, the *CT* algorithm is typically less stable (it describes noisy tracking trajectories) but is able to retrieve the target using the local and exhaustive searches described in **Chapter 2**. Concerning the *MS*+*CT* cooperation scheme, *MS* in general requires less cycles than *CT* but is only able to perform local searches. The combination method proposed here is able to automatically detect the sequences (and time intervals of frames along them) where each methods fits better, *e.g.*, in moving camera sequences the *CT* algorithm is more suitable while in static camera sequences the *MS* or *FoT* algorithms can be used as long as the target is not occluded. In future work, it is planned to propose a unified method which would combine the three tracking techniques all together to provide a full adaptation to the target and the context.

So far we have only talked about single object tracking applications in static and moving camera configurations. Next chapter introduces a multiple object tracking method that analyzes the dynamics of the targets and models their appearance using covariance matrices. Beforehand, some improvements to the covariance descriptor color and texture distinctiveness capabilities are proposed.

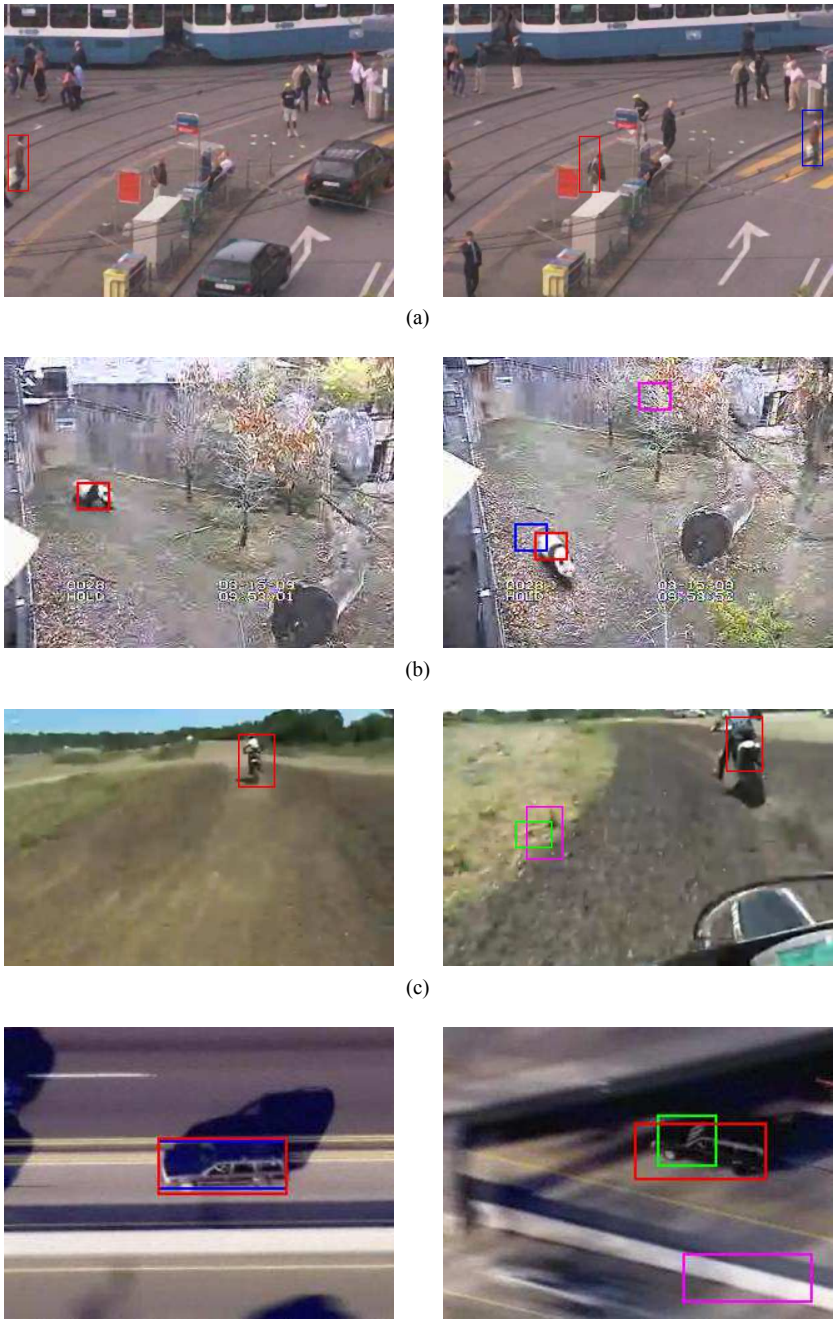


Figure 3-13: Sequences used in the experiments: first and last frames with the different tracking results displayed with different colors: MS with colorspace switching in pink, CT in red, and CT/MS in blue /green.

k=7484 k=7550 k=7574 k=7672

Figure 3-14: Tracking results of sequence *Pedxing*

k=1 k=80 k=190 k=300

k=400 k=800 k=900 k=940

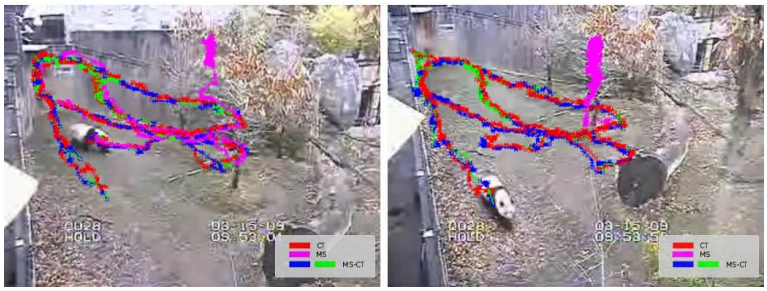


Figure 3-15: Tracking results of sequence *Panda*, and trajectories. The left image shows the forward trajectory, the right one shows the reverse trajectory.

k=1 k=50 k=100 k=150

k=200 k=220 k=300 k=401

Figure 3-16: Tracking results of sequence *Motocross*

CHAPTER 4

Robust multi-target covariance tracking and re-identificacion

Contents

Introduction

As stated in **Sections 1.2.5** and **2.2.4** covariance matrix representations offer a very practical alternative that allows us to build models based on local image correlations between different image cues (e.g., intensity, directional gradients, spatial coordinates, texture representations, optical flow, etc). Covariance matrices are compact and distinctive, these are highly estimated in tracking and matching applications. Even-though covariance matrices are compact, the computation time required to calculate distance between samples tends to be high and grows exponentially with the number of features. The reason is that the Riemannian distance computations require complicated matrix algebra operations. This limitation may have a direct impact on the required time for matching.

One of the objectives of this chapter is to find texture and feature operators and the best set of combinations that enhance the robustness and distinctiveness of the covariance descriptor while keeping its size as small as possible with the intention of minimizing not only the required storing space but the computation time. The new sets of texture and color features proposed are evaluated for three different applications with comparisons to *state of the art* methods. The set of texture

features studied here make use of the popular local binary patterns (*LBP*) (described in **Section 1.1.3**) proposing a new feature combination which is suitable to embedding it inside the covariance matrix descriptor and avoiding the problems that other naive similar approaches have found. In turn, the color features studied here are based on the color constancy theory and the Gaussian color models introduced in **Section 1.1.1**. The objective is to robustify the covariance descriptor by enabling it to handle illumination and color saturation changes using color invariants.

The other objective of the chapter is to introduce an original multiple-object tracking method that minimizes a combinatorial discrete energy function based on the likeliness of many different target trajectories and their congruence with their respective appearance models (based on covariance matrices). This algorithm is discussed at the end of the chapter in **Section 4.4.1**.

4.1 Discriminant texture information in covariance matrices

To improve their texture discriminant power Pang *et al*. proposed their *GRMC* (Gabor Region Covariance Matrix) which results from collecting the Gabor filter bank responses (view **Section 1.1.3**) into the set of features used to build a covariance matrix descriptor [104] and [135]. The inconvenience of *GRMC* is the large number of features it uses (each one for each filter). *GRMC* is not only expensive to compute but to match. Some publications replace Gabor filters and include local binary patterns: the local binary covariance matrix (*LBCM*) of Guo and Ruan [59] and the Gabor-LBP based region covariance descriptor (*GLRCD*) of Zhang and Li [154]) are just two examples.

In the rest of this section an *enhanced* local binary covariance matrix descriptor (*ELBCM*), after this, it is evaluated for three different applications: texture classification, facial expression classification and object tracking.

4.1.1 Enhanced local binary covariance matrices (*ELBCM*)

The right way to introduce textural information in the form of local binary patterns into the covariance matrix descriptor is not obvious. *GLRCD* uses a direct embedding of the *LBP*'s decimal value which can be very unstable for the case of local neighbourhood rotations. Additionally, carrying typical arithmetic operations with such values is meaningless (*i.e.* adding or averaging two or more *LBP* decimal values does not mean anything in terms of texture). *LBCM* uses each bit in the LBP pattern as an independent feature forming a string of P bits. This is more stable and doing arithmetic independently for each bit in the string is well defined. The problem is that the number of features inside the covariance matrix grows with the number of P bits considered by the $LBP_{P,R}$ operator. This has a significant impact on the execution speed. To improve this situation, an enhanced *LBP*-based covariance descriptor (*ELBCM*) is proposed in this section, the *ELBCM* descriptor uses the angles defined by the uniform *LBP* patterns presented in **Section 1.1.3**.

In the case of the $LBP_{8,R}$ operator there are 58 possible uniform patterns, but the strings formed by all ones and all zeros are ignored because their angle representations are the same, which can can be misleading. Figure 4-1 shows the remaining 56 different patterns used to construct our *ELBCM* covariance descriptor when this operator is applied. The starting and ending angles θ_0 and θ_1 are marked by the curved arrows inside each pattern. Since computations have to be made on circular quantities, the angles are converted to points on the unit circle, e.g., θ_0 is converted to $(cos\theta_0, sin\theta_0)$.

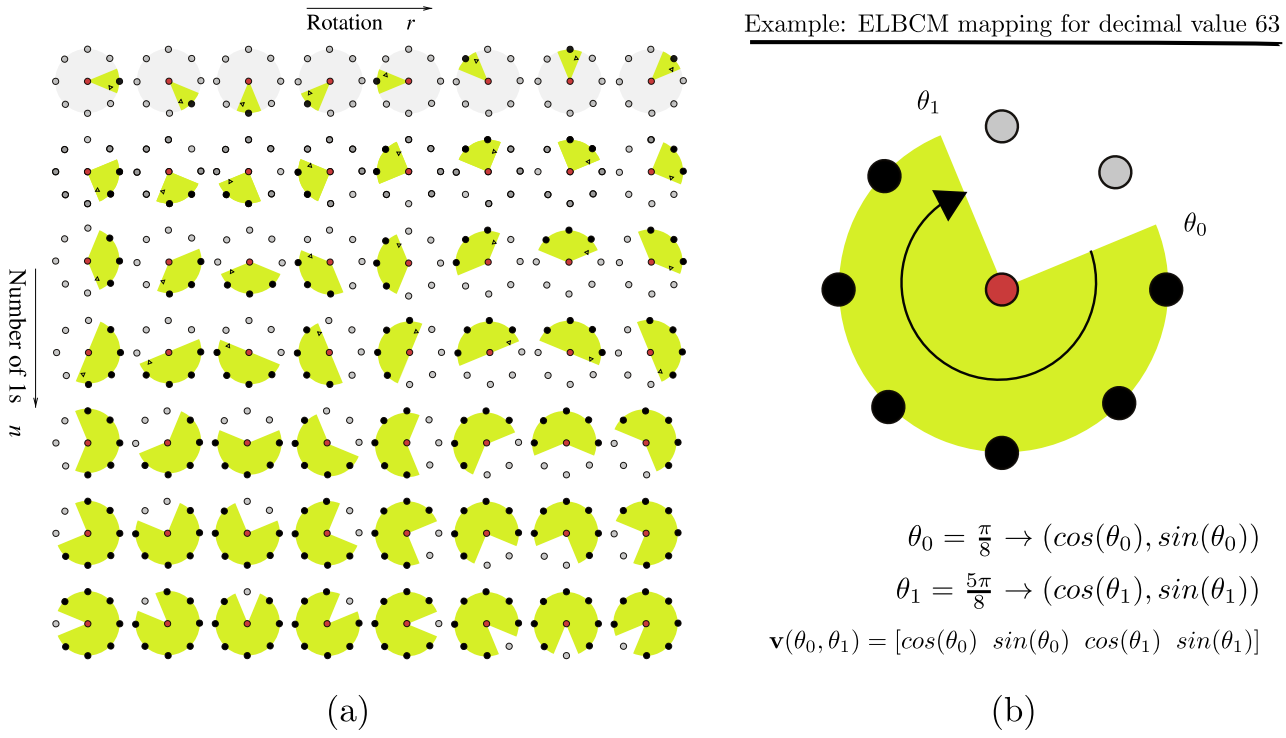


Figure 4-1: (a)The set of 56 different uniform *LBP* patterns in a $(8, R)$ neighborhood. For each pattern a curved arrow indicates the start and end angles (θ_0 and θ_1). (b)In *ELBCM*, the uniform *LBP* pattern defined by two angles θ_0 and θ_1 is described by $\mathbf{v}(\theta_0, \theta_1) = [cos(\theta_0) \;\; sin(\theta_0) \;\; cos(\theta_1) \;\; sin(\theta_1)]$. The *LBP* pattern represented by decimal value 63 maps to $\mathbf{v}(\pi/8, 5\pi/8) = [cos(\pi/8) \;\; sin(\pi/8) \;\; cos(5\pi/8) \;\; sin(5\pi/8)]$.

The vector

$$\mathbf{v}(\theta_0, \theta_1) = [cos(\theta_0) \;\; sin(\theta_0) \;\; cos(\theta_1) \;\; sin(\theta_1)] \tag{4.1}$$

uniquely describes each one of the considered *ULBP* patterns as a function of (θ_0, θ_1). An example for calculating vector $\mathbf{v}(\theta_0, \theta_1)$ is provided in Figure 4-1-(b) where the decimal value 63 (bit string 00111111) is mapped to the angles $\theta_0 = \pi/8$ and $\theta_1 = 5\pi/8$ (angles are measured anti-clock wise from the x-axis as usual). The enhanced local binary covariance matrix (*ELBCM*) texture descriptor is built using the mapping function

$$\phi(I, x, y) = \left[\begin{array}{cccc} x & y & I(x,y) & \mathbf{v}(LBP(x,y)) \end{array} \right]. \tag{4.2}$$

The $LBP(x, y)$ operator provides the pair of angles (θ_0, θ_1) at (x, y) to $\mathbf{v}$. Local neighbourhoods with non-uniform *LBP* patterns are ignored and their feature vectors do not contribute to the covariance matrix computation. Because $\mathbf{v}$ is a four dimensional vector, the total dimension of $\phi(I, x, y)$ is $d = 7$ (the resulting covariance matrices are of size 7×7).

The use of trigonometric formulas in (4.2) should make *ELBCM* computation more time consuming than the other LBP-based methods, but this problem completely overcome using a look-up table that maps directly the 56 *ULBP* decimal numbers to the sine and cosine values of θ_0 and θ_1.

ELBCM has multiple advantages in comparison to previous covariance-based texture descriptors. It is more compact (7 components of *ELBCM* vs. 11 components of *LBCM*), more stable *i.e.* it is less affected by small rotations that reflect as changes on the angles θ_0 and θ_1 described by the *ULBP* pattern while for *LBCM* the same rotations irregularly affect the

value of the bits in $LBP(x, y)$ feature vector depending on their position. An example describing this behavior is presented in **Figure 4-2**. An additional problem of *LBCM* is observed in practice where non-positive-definite matrices appear more frequently. A possible explanation for this phenomenon is that bit-strings in the $LBP(x, y)$ operator can be very sparse in smooth regions. An additional advantage of the *ELBCM* descriptor is that its size is completely independent of the number of P neighbors used in the *LBP* pattern operator. This property can be very important for many different texture analysis applications such as classification, facial expression recognition and tracking. The *ELBCM* descriptor is evaluated for a variety of tasks in **Section 4.1.2** to confirm its superiority in comparison to other covariance-based configurations and to compare its performance against other state-of-the-art methods.

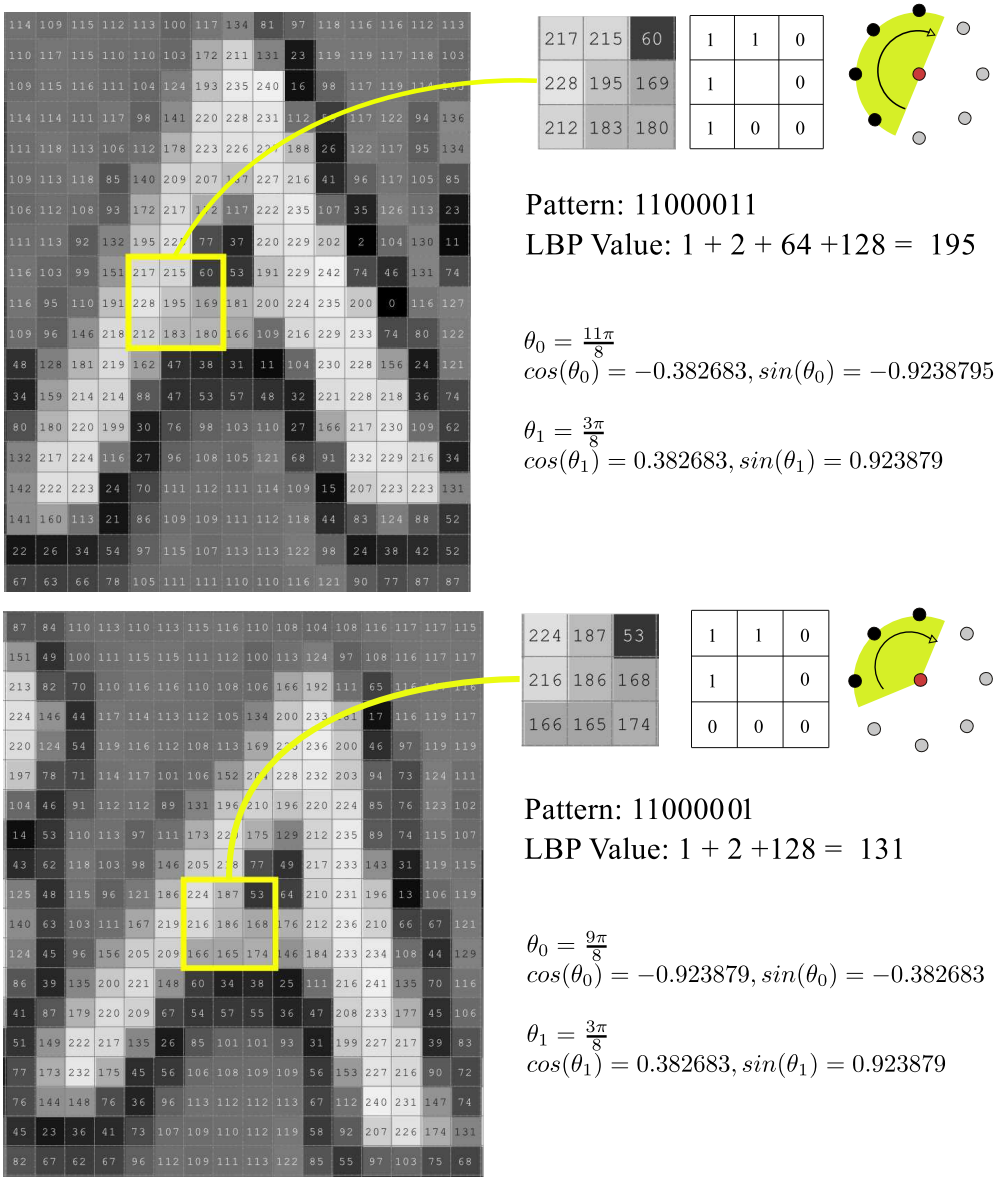


Figure 4-2: Example of the type of problems observed when LBP patterns are embedded inside covariance matrices. When a template is transformed by a linear transformation such as a small rotation, many of its 3×3 neighborhoods may be altered and the LBP pattern strings or decimal values are severely affected. In contrast, the proposed angular representation used to build the *ELBCM* descriptor is more stable.

The descriptive abilities of out *ELBCM* descriptor are evaluated for three different applications: texture and facial expression classification and tracking. Two different datasets are used for texture classification: the Brodatz and KTH-TIPS. The classification accuracy provided by *ELBCM* for these datasets is compared against results obtained by the following state-of-the-art methods: VZ-join, Lazebnik, Hayman, Tuzel, HLSS, Jalba, L^2ECM and *LBCM*.

4.1.2 *ELBCM* evaluation

In this subsection, we exhibit some possible applications of our *ELBCM* descriptor such as texture and facial expression classification and object tracking.

4.1.2.1 Texture classification

For performance evaluation we used two widely used datasets in texture classification. The Brodatz dataset and KTH-TIPS dataset [62].

Brodatz Dataset: The Brodatz dataset contains 111 different textures each one represented by a single 640×640 image. Scale, viewpoint or illumination changes are not a problem in this dataset, the main difficulty of this dataset in terms of classification is that it includes non-homogeneous textures, **Figure 4-3** displays some texture samples taken from this dataset. To have comparable results, we have followed the same methodology of [82] where each image is subdivided into 49 overlapping blocks of size 160×160 using a pixel block stride of 80. Each class is trained using twenty four randomly selected images from their respective 49 images set (the remaining ones are used for testing). During training, each image is now divided into four 80×80 patches and their *ELBCM* covariances are then computed. During testing, each covariance matrix of the testing image is compared to all the covariance matrices inside the training set using the Riemannian metric proposed in 2.30. A label is assigned to each test image using a KNN algorithm ($k = 5$) which counts the number of votes gathered by the four test image covariance matrices for each class. In Figure 4-4 the classification accuracy results for the Brodatz dataset are reported. Different feature map configurations in the covariance matrices were tested: Tuzel [139], *LBCM* [59] and *ELBCM*, other state-of-the-art results are included too (taken from [82]). The reported results are the average of twenty different experiment runs.

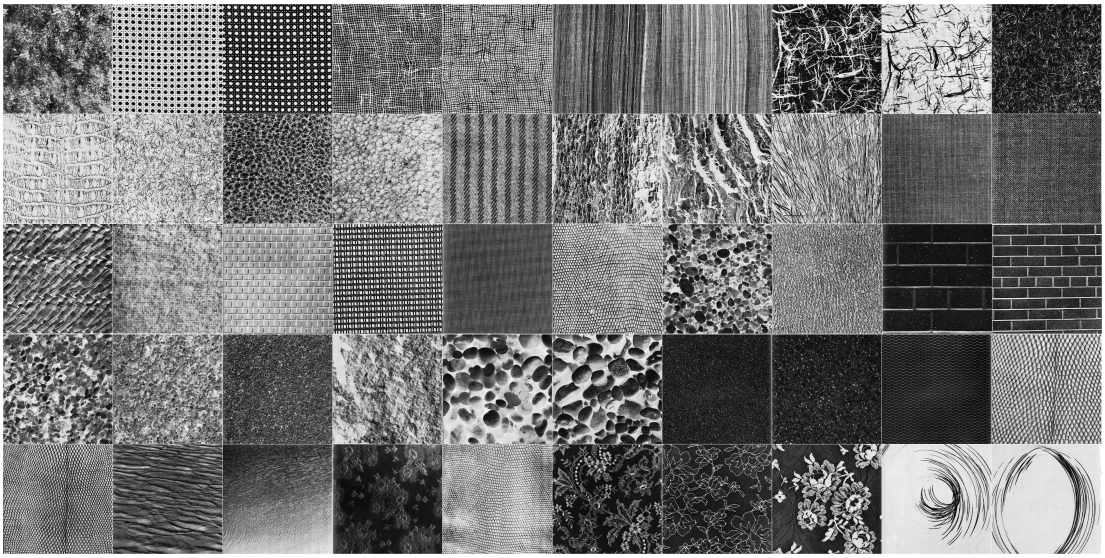


Figure 4-3: Some texture examples taken from the Brodatz dataset.

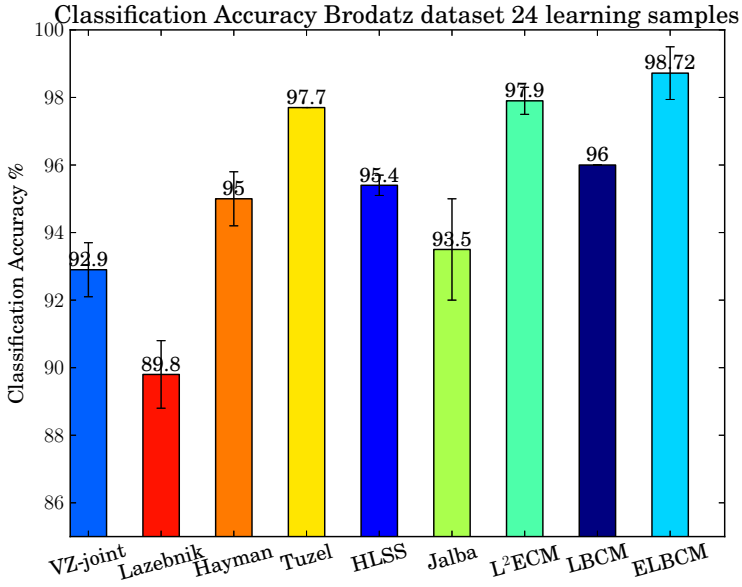


Figure 4-4: Texture classification accuracy for the Brodatz dataset.

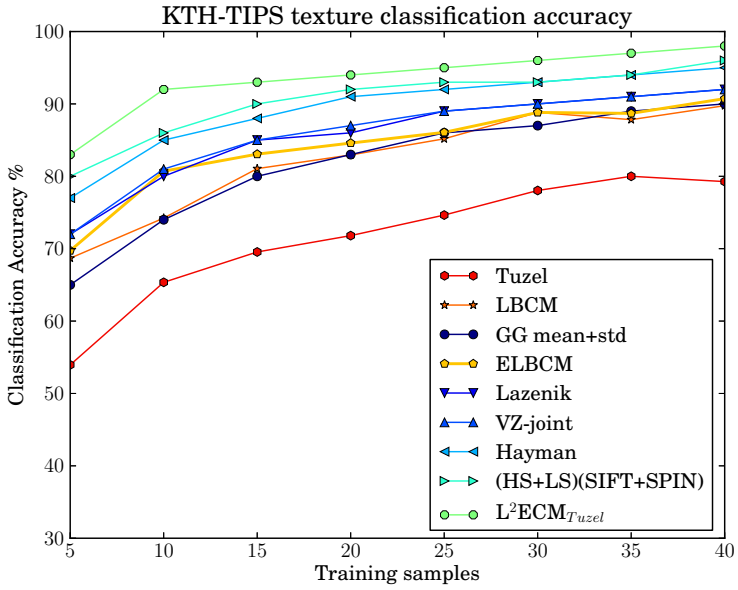


Figure 4-5: Texture classification accuracy for the KTH-TIPS dataset.

KTH-TIPS Dataset: This dataset [62] is composed by ten different texture classes represented by 81 different samples. This dataset is more challenging than Brodatz because here each texture class is represented by images taken at different scales, illumination and pose. The size of the samples is 200×200 pixels. Similar to the previous experiment each image is uniformly subdivided into four blocks and a covariance matrix is computed on each one. Figure 4-5 shows the classification accuracy depending of each feature combination or method depending on the number of training images. *ELBCM* surpasses other methods based on simple covariance descriptors, but others methods which are designed specifically to handle scale changes obtain better results (e.g. $(HS + LS) + (SIFT + SPIN)$). To our knowledge, L^2ECM_{Tuzel} [82] remains as the best method for this dataset.

4.1.2.2 Facial expression classification

To evaluate the performance of our covariance descriptor in the facial expression recognition problem we used the JAFFE dataset[1] which consists of 213 images from Japanese female individuals, covering seven expression categories (neutral, anger, disgust, fear, happiness, sadness ans surprise). To compare with [59] we followed their same methodology: images were automatically cropped to align the eyes, then they were resized to 100×100 pixels and 21 samples of each expression were used for training and only one training sample for testing. The experiments were repeated several times and their averages are displayed in Figure 4-7. *ELBCM* is followed by *Gabor GRCM* [154] and $LBCM_{(8,1)}$ [135] as the best adapted descriptors for this problem. Tuzel's descriptor [139] offers very poor performance for this dataset. It was not possible to compare *ELBCM* to the other methods used in the texture classification experiments as results for this dataset are not publicly available.

[1] http://www.kasrl.org/jaffe.html

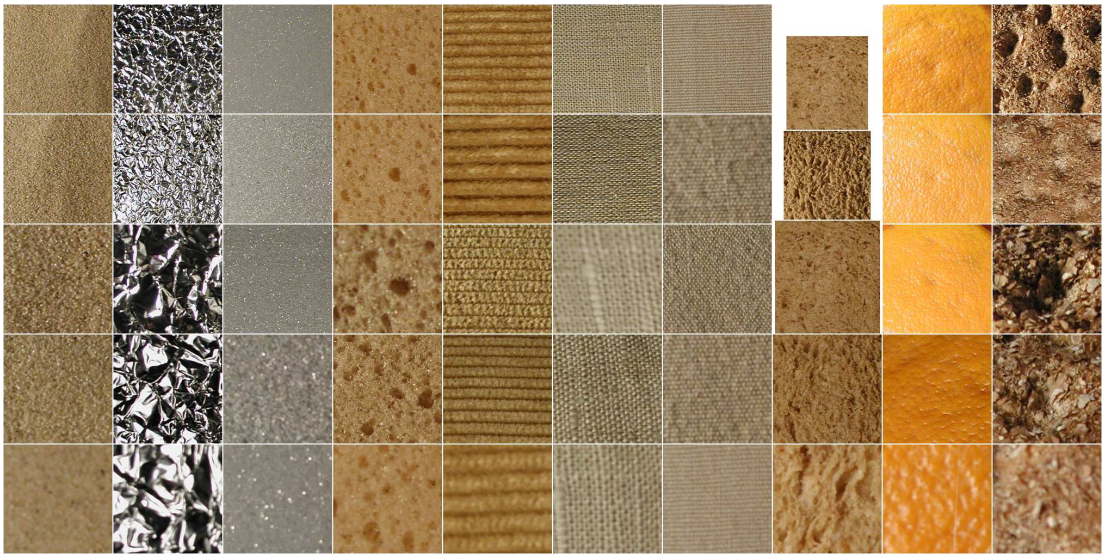


Figure 4-6: Some texture examples taken from the KTH-TIPS dataset.

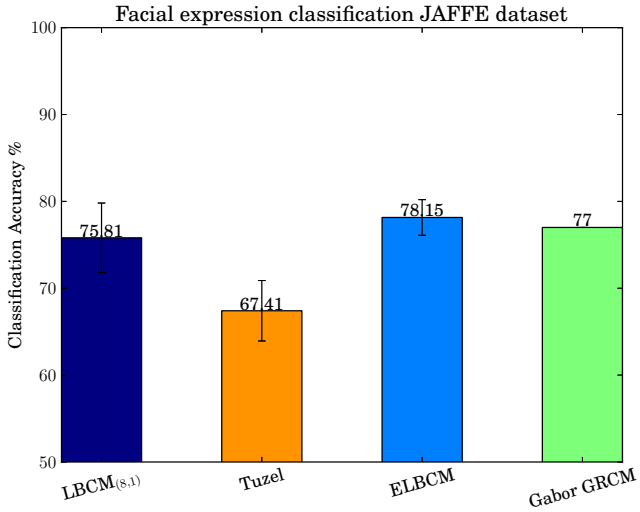


Figure 4-7: Facial expression classification results for the JAFFE database using 21 learning samples.

4.1.2.3 Object tracking

Using the covariance matrix as a global region descriptor, we compare *ELBCM* descriptors with other configurations: Tuzel's [139], *LBCM* [135] and *LRCD* [154] (LBP decimal value without Gabor filters configuration). The initial trackers position is marked in the first frame by hand. We implemented a very simple tracker that is limited to fixed-scale targets and which follows targets locally (i.e. if targets are located far from the trackers positions they are not found) testing different random locations taken from a 2D Gaussian distribution centered at the target's previous location and variances proportional to the target dimensions. Using the similarity of the tested locations against the target model as weights, a weighted average is calculated to estimate the current target location. Three different sequences are discussed here and

what interest us is the precision at which the different feature combinations follow the targets and their tracking times. When ground-truth information is available precision is measured using following weighted error

$$w_{error} = \frac{\|C_{gt} - C_{tracked}\|}{R} \tag{4.3}$$

where C_{gt} and $C_{tracked}$ are the ground truth and tracked centers respectively, and R is the target size, calculated as the average of the vertical and horizontal bounding box lengths.

- **Jumping sequence**: This sequence (313 frames, size: 352×288) appears in gray-scale, it shows a guy jumping rope in a garden, a building and some trees are behind. The object of interest is the guys face. Due to the jumping speed and random camera movements images appear extremely blurred. After frame $t = 41$ Tuzel and *LRCD* confuse the guys face with the background losing the target and never being able to re-catch it again. In contrast, *LBCM* and *ELBCM* follow the target until the end. In some frames and due to the shaking of the camera, the guys face appears extremely blurred and trackers drift considerably from the real location, but *LBCM* and *ELBCM* are able to catch it again as soon as the target appears reasonably clear again. This sequence comes from the TLD dataset [75]. Results for this sequence are summarized on the first column of **Table 4.1**, because *LRCD* matrices are the smallest (4×4) they are the fastest ones but less precise. The best descriptor in terms of performance is *ELBCM* which is two times faster than *LBCM* and more precise in terms of the w_{error}.

- **Panda sequence**: This sequence (3000 frames, size: 312×233) was filmed in color and comes from TLD dataset too [75]. It shows a panda enclosed in a park. At the beginning of the sequence the camera zooms in and out while the panda moves. The panda is a non-rigid target and its appearance changes a lot throughout the sequence: it turns on itself and its shape and color distribution drastically change. In addition, the panda passes behind a tree, and the background changes. All methods follow the target correctly until frame $t = 1001$ when it is no longer visible, then it re-enters to the scene at $t = 1182$ *LRCD*, *LBCM* and *ELBCM* are able to re-identify it and track it until frame $t = 2656$ when the camera zooms in and the target size changes significantly affecting the appearance, all these events are observable in Figure 4-11 and 3-11a. Execution time and w_{error} results are shown on the second column of **Table 4.1**. While *LRCD* is very fast it oscillates a lot, in turn, *ELBCM* offers a good balance of precision and speed.

- **Pedxing 3 sequence**: Used in [81], this sequence (188 frames long from $t = 7484$ to $t = 7672$, size: 640×480) was registered in color. The target is a pedestrian who is crossing the street, and passes behind a panel. Due to compression noise and other artifacts the quality of the image is very poor. All methods but *ELBCM* and *LBCM* fail to catch the pedestrian after he passes behind the panel. Tracking results are given on the third column of **Table 4.1**. There is no publicly available ground-truth information for this target, so the target was labeled using Vatic[2] [145] to offer a similar comparison to the other sequences, the resulting ground-truth file can be downloaded from my personal website[3].

[2] Video Annotation Tool from Irvine, California (VATIC) `http://web.mit.edu/vondrick/vatic/`.
[3] `http://andresromero.github.io/`

Figure 4-8 and **4-9** show some samples taken each five frames testing the four vector configurations, **Figure 4-8** corresponds to the *Jumping* sequence, here, the shakings of the camera, the nonrigid deformations on the face and the blurring conditions are very evident. While the tracker at some points seems to start losing the target, the distinctiveness of the covariance model allows our method to recover it. The same for the *Panda* sequence in **Figure 4-9a** where the target shows noticeable changes of size, perspective and other non rigid transformations, nevertheless the *ELBCM* covariance tracker is able to recover it even when it gets out and re-enters the scene. The third tested sequence was *Pedxing3*, the set of cropped bounding boxes in **Figure 4-9b** denote the non-rigid deformations experienced by the target and the occlusion that occurs when the target passes behind the panel, the tracker was able to recover the target and follow it until he gets out of the. **Figures 4-10, 4-11** and **4-12** show some frames taken from each one of the tested sequences, bounding boxes in yellow correspond to the $ELBCM_{Lum}$ feature vector configuration, $LBCM$ is denoted in red, Tuzel in blue and the *LRCD* in green. Plots of the w_{error} for each of the tracking sequences are provided too. In the jumping sequence, the performance offered by *LBCM* and $ELBCM_{Lum}$ are comparable, Tuzel and *LRCD* both fail (the w_{error} grows and never recovers for them after $t = 50$). For the *Panda* sequence, all the feature configurations behave more or less the same before the target gets out of the scene around frame $t = 1000$. When the target re-enters around $t = 1300$, all feature configurations but Tuzel are able to recover it, the target is followed correctly until the camera zooms in (around $t = 2450$) here only *ELBCM* is able to recover the target and follow it until the end of the sequence $t = 3000$.

Figure 4-8: Bounding boxes cropped from the *Jumping* sequence.

(a) *Panda* sequence.

(b) *Pedxing3* sequence.

Figure 4-9: Bounding boxes cropped from the *Panda* and *Pedxing3* sequences.

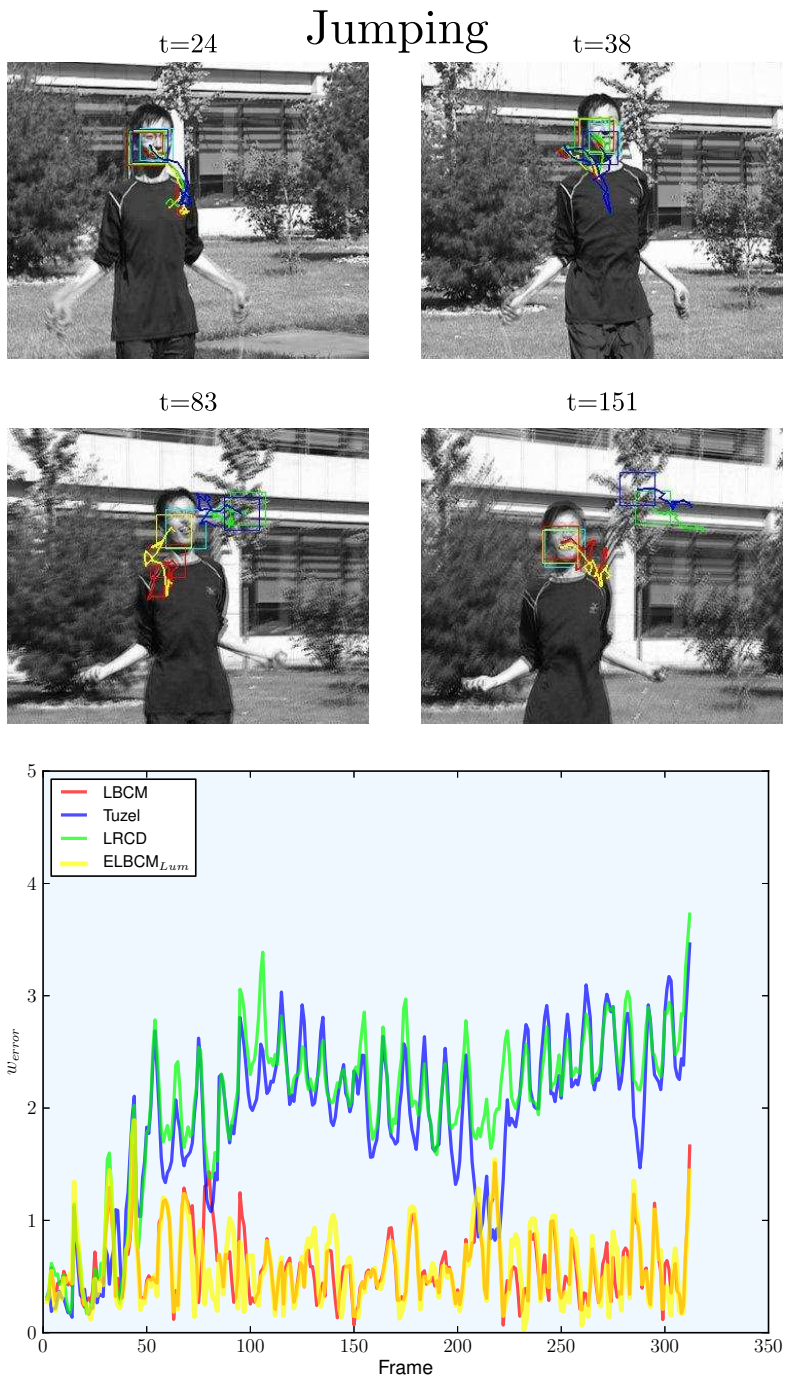


Figure 4-10: Tracking results in the Jumping sequence. Tuzel's (blue) tracker and *LRCD* (green) get stuck in the background after the first frames. *LBCM* (red) is less precise than *ELBCM* (yellow) but both follow the target until the end. Ground-truth information is coloured in cyan.

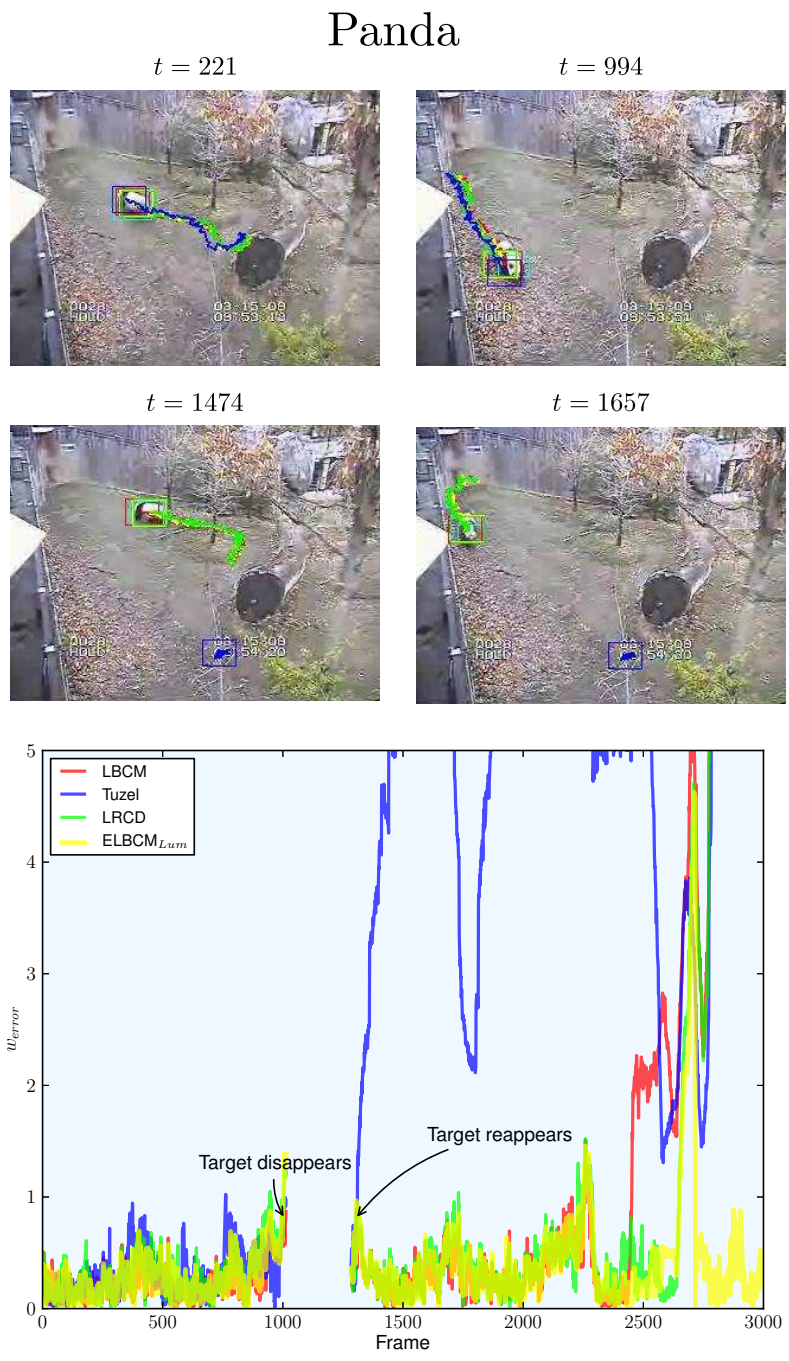


Figure 4-11: Tracking results in the Panda sequence. During the first 1000 frames of the sequence all targets follow correctly the target. Tuzel's tracker (blue) loses the target when it gets out of the image. When it re-enters, *LRCD* (green), *LBCM* (red) and *ELBCM* (yellow) follow the target correctly until $t = 2656$ when the camera zooms in and the object appearance changes significantly. Ground-truth information is colored in cyan.

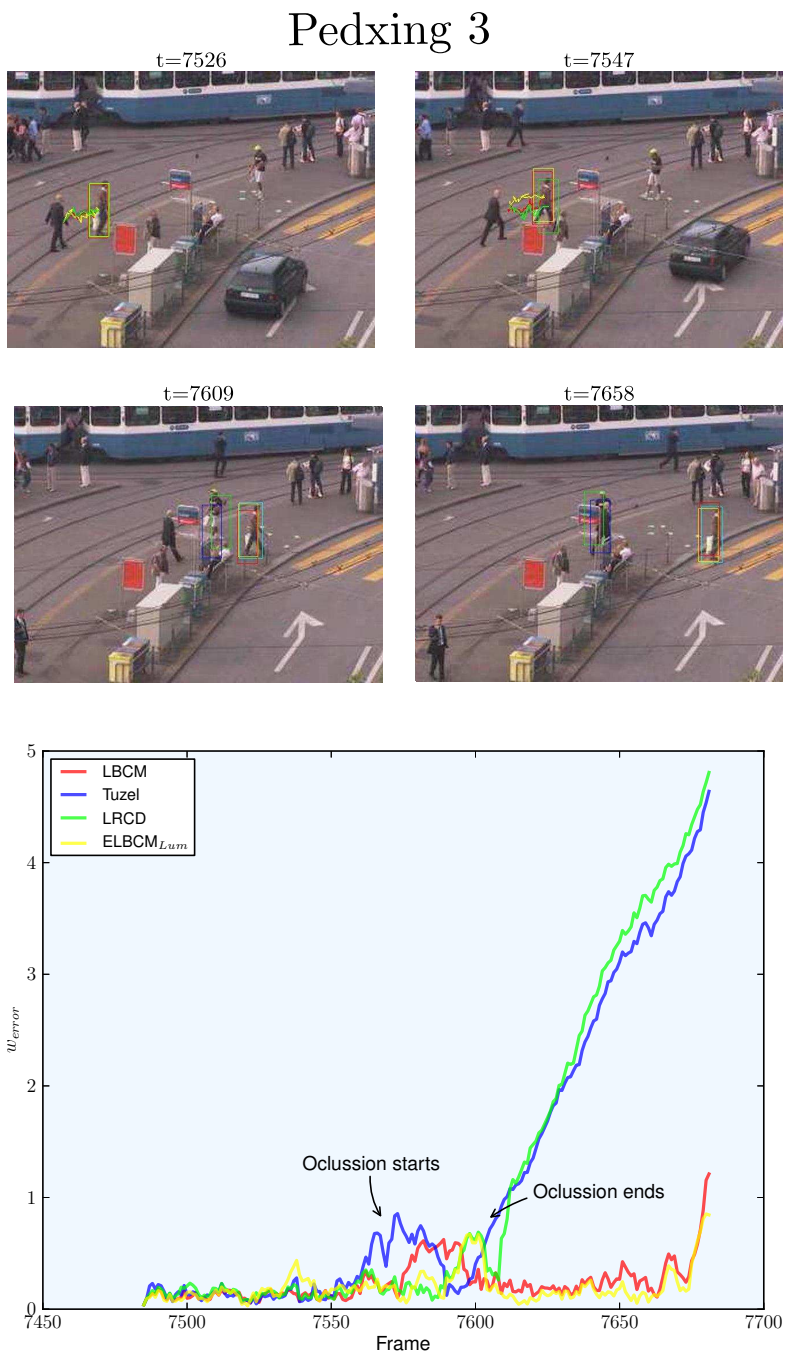


Figure 4-12: Tracking results for Pedxing3 sequence. Tuzel's tracker (blue), *LRCD* (green) and *LBCM* (red) lose the target when it passes behind the panel at the center of the image. Only *ELBCM* (yellow) and *LBCM* (red) succeed to follow the target until the end. Ground-truth information is colored in cyan.

Table 4.1: Tracking time and accuracy for three different sequences and four different feature combinations.

Features	Jumping		Panda		Pedxing3	
	time/frame [ms]	w_{error}	time/frame [ms]	w_{error}	time/frame [ms]	w_{error}
Tuzel	4.83	1.83	3.65	1.93	20.5	-
LBCM	10.3	0.36	7.57	0.28	40.6	-
ELBCM	4.82	0.26	3.65	0.27	20.5	-
LRCD	2.25	2.35	1.71	0.38	6.69	-

4.2 Discriminant color information in covariance matrices

The relevance of color to improve the discriminant power of the covariance descriptor is discussed and evaluated here. Many feature combinations that include color inside the covariance descriptor are possible, the most simple approach just replaces the luminance component with the individual color channels (*e.g.*, R, G and B) or by a color invariant. To avoid increasing unnecessarily the number of features used to construct the covariance matrix, spatial-gradients are not computed for each color channel individually, the luminance gradient or texture operator is preferred instead as described in the previous section.

Four different feature combinations are used to represent color information inside the covariance matrices: RGB, HSV, Guassian color models discussed in **Section 1-4** and by the $L1$ invariant presented in **Section 3.1.1**. Thus the set of colors to test is $\{RGB, HSV, Gaussian, L1\}$. Gradient or textural information is given by the luminance gradients (I_x and I_y) or by the local binary operator used by *ELBCM*. **Table 4.2** shows the resulting list of feature combinations. As in the previous section, the performance offered by the covariance descriptors on this table was evaluated for texture classification and object tracking applications.

Table 4.2: Feature combinations.

Color space	Feature set $\phi(I, x, y)$
RGB_{grads}	$\left[\begin{array}{ccccccc} x & y & R & G & B & \lvert I_x\rvert & \lvert I_y\rvert \end{array}\right]$
HSV_{grads}	$\left[\begin{array}{ccccccc} x & y & H & S & V & \lvert I_x\rvert & \lvert I_y\rvert \end{array}\right]$
Gauss_{grads}	$\left[\begin{array}{ccccccc} x & y & \hat{E} & \hat{E}_\lambda & \hat{E}_{\lambda\lambda} & \lvert I_x\rvert & \lvert I_y\rvert \end{array}\right]$
L1_{grads}	$\left[\begin{array}{ccccc} x & y & L1 & \lvert I_x\rvert & \lvert I_y\rvert \end{array}\right]$
$ELBCM_{RGB}$	$\left[\begin{array}{ccccccccc} x & y & R & G & B & cos(\theta_0) & sin(\theta_0) & cos(\theta_1) & sin(\theta_1) \end{array}\right]$
$ELBCM_{HSV}$	$\left[\begin{array}{ccccccccc} x & y & H & S & V & cos(\theta_0) & sin(\theta_0) & cos(\theta_1) & sin(\theta_1) \end{array}\right]$
$ELBCM_{Gauss}$	$\left[\begin{array}{ccccccccc} x & y & \hat{E} & \hat{E}_\lambda & \hat{E}_{\lambda\lambda} & cos(\theta_0) & sin(\theta_0) & cos(\theta_1) & sin(\theta_1) \end{array}\right]$
$ELBCM_{L1}$	$\left[\begin{array}{ccccccc} x & y & L1 & cos(\theta_0) & sin(\theta_0) & cos(\theta_1) & sin(\theta_1) \end{array}\right]$

4.2.1 Texture classificacion using color-texture covariance matrices

Here we repeat our texture classification tests using the color feature combinations of **Table 4.2**. Unfortunately the Brodatz and the JAFFE datasets used in the previous section contain gray-scale images only. So, only the KTH-Tips dataset was used here.

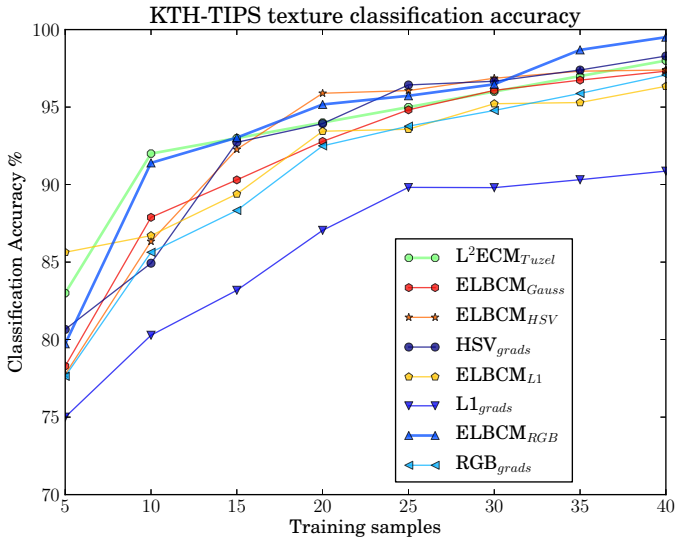


Figure 4-13: Texture classification accuracy for the KTH-TIPS dataset using color-texture covariance matrices.

Figure 4-13 shows the classification accuracy that was obtained for this dataset using all the enlisted feature combinations, it also includes the accuracy results for the L^2ECM_{Tuzel} features of [82] which was the best feature combination for the non-color tests of the previous section (see **Figure 4-5**). The $L1_{grads}$ feature set is clearly inferior when compared to other color feature combinations for this dataset. However, it still obtains better results that the Tuzel feature combination and it is comparable to the LBP-based $ELBCM_{Lum}$ results of the previous section (see **Figure 4-5**). Mixing the LBP-based texture information with the L1 color invariant clearly improves the results as the $ELBCM_{L1}$ feature set behaves much more better than $L1_{grads}$ and obtains slightly better results than the RGB_{grads}. Among the feature sets based on the gradients, HSV_{grads} is the one which obtains the best results. In general $ELBCM_{HSV}$ and $ELBCM_{RGB}$ get the best results, they are comparable to the state-of-the-art results offered by L^2ECM_{Tuzel}.

4.2.2 Tracking using color-texture covariance matrices

Two sequences were evaluated here, they analyze two extreme cases, one where the target is highly saturated and where its color is very distinctive in comparison to the environment (*i.e.* the background and other people around the target), and a second sequence where the target initially appears low-saturated and where lighting changes occur altering significantly the appearance of the target.

- **Pets 2009 sequence**: The effect of color in the covariance descriptor was verified by selecting a highly colored pedestrian from the **PETS2009-S2-L1-View01**. The target is a girl wearing a highly saturated red jacket. Color here proves to be valuable as the pedestrian traverses many obstacles, crosses other pedestrians along its path, leaves and later re-enters the scene. **Figure 4-14** shows the evolution of the target throughout the sequence. **Figure 4-15-(a)** shows the locations of the trackers at $t = 466$, at this point all the feature combinations have succeeded to follow target throughout its first crossing. **Figure 4-15-(b)** depicts the positions of the trackers at $t = 750$, most of the trackers have already drifted from the target, only $ELBCM_{Gauss}$, $ELBCM_{RGB}$ and RGB_{grads} have succeeded to track the target consistently throughout the whole sequence. An increased level of invariance typically leads to a reduction of the distinctiveness of the descriptor, this may explain why the L1-based descriptors rapidly fail (around $t = 450$), then HSV-based descriptors ($ELBCM_{HSV}$ and HSV_{grads}) fail around $t = 710$ when the target crosses a different pedestrian wearing a blue jacket with some reddish bands. Only the RGB_{grads}, $ELBCM_{RGB}$ and $ELBCM_{Gauss}$ succeed in tracking the target until the sequence end, the *ELBCM*-based descriptors are slightly more precise than the gradients based RGB_{grads}.

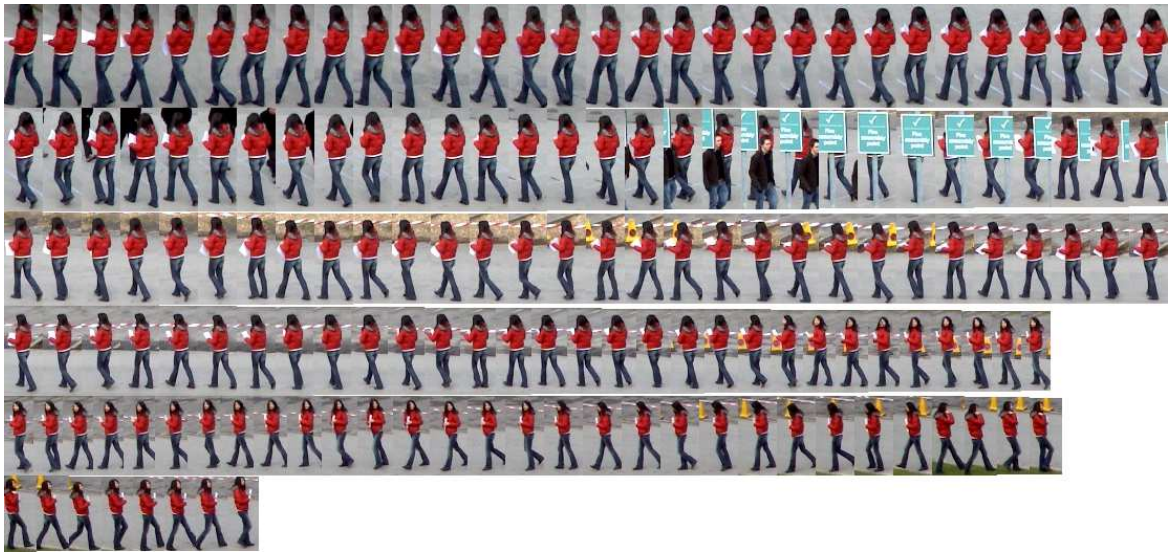
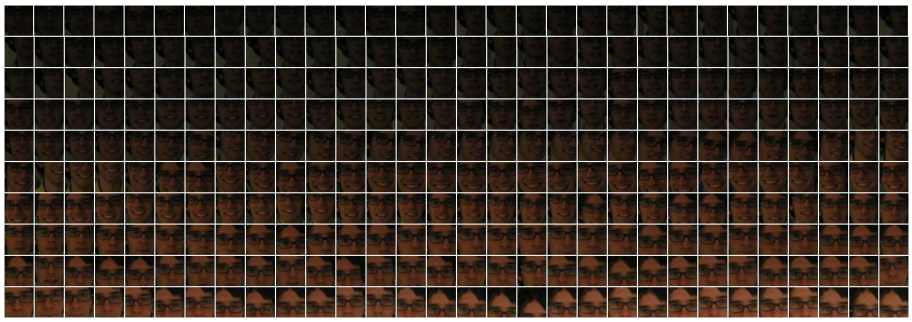


Figure 4-14: Evolution on the appearance of the selected target on the **PETS2009-S2-L1-View001** sequence.

- **David sequence**: This sequence (722 frames, size: 320×240) appears in color, it is an indoors sequence showing a guy walking in front of the camera. The object of interest is the guys face and the difficult part is to handle the lighting changes that occur as the guy moves. The evolution of the appearance of the target is shown in **Figure 4-17-(a)**. **Figures 4-17-(b)** and **(c)** show some frames taken from this sequence at times $t = 5$ and $t = 578$. The precision error w_{error} is plotted in **Figure 4-18**. Among all the feature combinations in **Table 4.2** only L1_{grads} and $ELBCM_{L1}$ resist the effects that the illumination changes have on the appearance of the target. L1_{grads} follows the target throughout the sequence while $ELBCM_{L1}$ loses it before $t = 100$ and recovers it many frames after (about $t = 520$) the reason is that low illumination produces few valid LBP for the *ELBCM* descriptor. The evolution of the precision error w_{error} for all the feature combinations is plotted by the curves in **Figure 4-18**.

(a) (b)

Figure 4-15: (a) and (b) show the location of the target at $t = 469$ and $t = 750$ respectively.



(a)

(b) (c)

Figure 4-17: (a)Shows the evolution on the appearance of the target throughout the sequence. Due to lighting changes, the appearance of the target changes drastically, (b) and (c) show the appearance at $t = 5$ and $t = 520$ respectively.

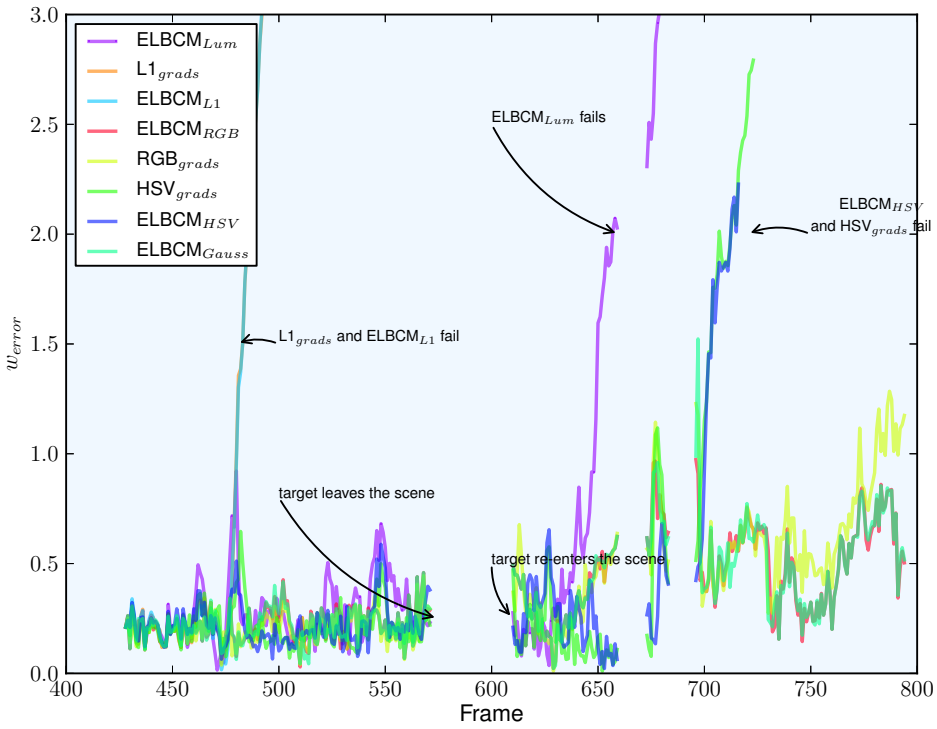


Figure 4-16: Tracking precision error w_{error} obtained using all the color-based feature set combinations for *PETS2009-S2-L1-View001* sequence.

4.3 Target re-identification with covariance matrices

The multiple target tracking problem and its relatives (e.g., trajectory analysis, activity recognition) are even more difficult when targets have to be detected and re-identified from a second camera and with considerable differences in illumination and perspective and elongated separations in time.

Background clutter, appearance instabilities, illumination and scale changes are just some of the typical difficulties found by the existing algorithms that have tried to solve this problem. In addition, when targets have non rigid motion or low textural and structural contents the gradient or corner-based methods such as the classical KLT [8] or SIFT [89] are not appropriate. Kernel-based methods like Mean-shift [30] are usually well adapted to such objects since they rely on global statistical distributions. The price to pay is a decrease of discriminant power, therefore several attempts have enhanced the method by background subtraction [70], color-space switch [80] and by using a spatio-colorimetric histogram [16]. Bak *et al*. proposed in [6] a dense grid of covariance matrices to handle these problems [6] together with the so called *Mean Riemannian Covariance* (*MRC*) matrices which are used to blend the appearance information from multiple samples.

Given a set of N covariance matrices $\{C_1, C_2, \cdots, C_N - 1\}$, the Karcher or Fréchet mean, is the value μ which minimizes the set of squared distances

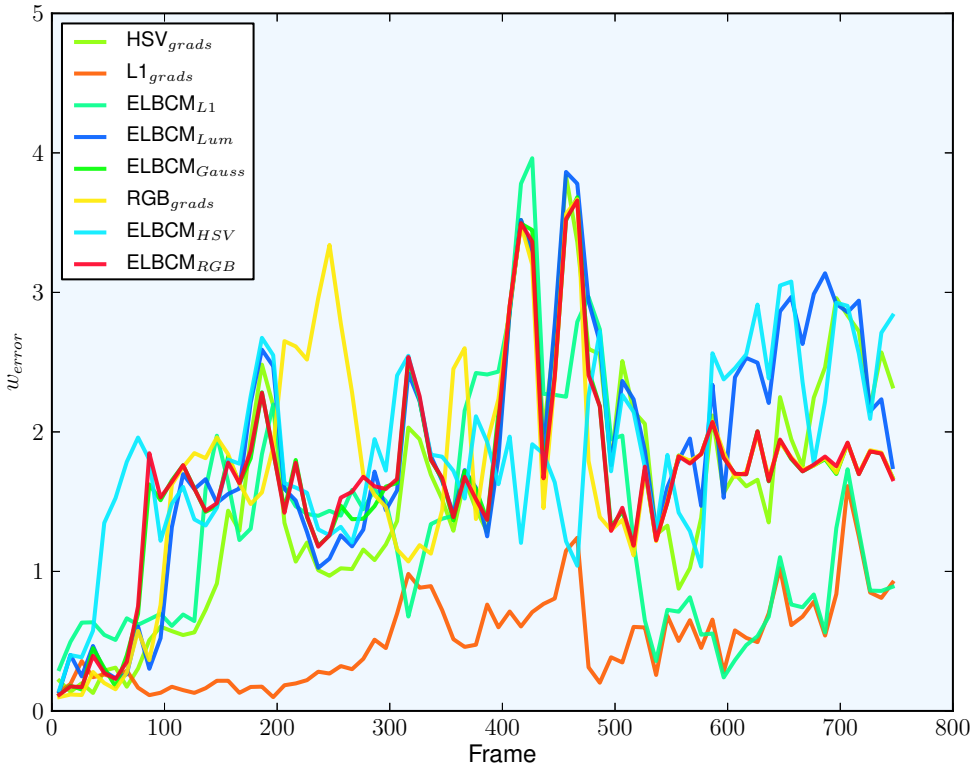


Figure 4-18: Tracking precision error w_{error} obtained using all the color-based feature set combinations for *David* sequence. Due to the illumination changes, only the L1-based color feature sets (L1$_{grads}$ and *ELBCM*$_{L1}$) succeed to track the target.

$$\mu = arg \min_{C \in \mathcal{M}} \sum_{i=1}^{N} \rho^2(C, C_i), \tag{4.4}$$

the value of μ is calculated iteratively following a Newton gradient descent method adapted for Riemannian manifolds. The approximate value of μ at step $t+1$ is

$$\mu_{t+1} = exp_{\mu_t} \left[\frac{1}{N} \sum_{i=1}^{N} log_{\mu_t}(C_i) \right], \tag{4.5}$$

where, exp_{μ_t} and log_{μ_t} are specific operators uniquely defined on the Riemannian manifold. Equations (4.6) and (4.7) express how to calculate them

$$Y = exp_X(W) = X^{\frac{1}{2}} exp(W) X^{\frac{1}{2}}, \tag{4.6}$$

$$Y = log_X(W) = log(X^{-\frac{1}{2}} W X^{-\frac{1}{2}}). \tag{4.7}$$

Bak *et al*. achieve great re-identification rates using a dense grid of of *MRC* matrices. For the case of human signatures, each image is scaled into a fixed size of 64×192 pixels where a grid of overlapping 16×16 pixel size cells is constructed. Neighboring cells are separated by 8 pixel steps. In total, 161 *MRC* are used to construct the human signature.

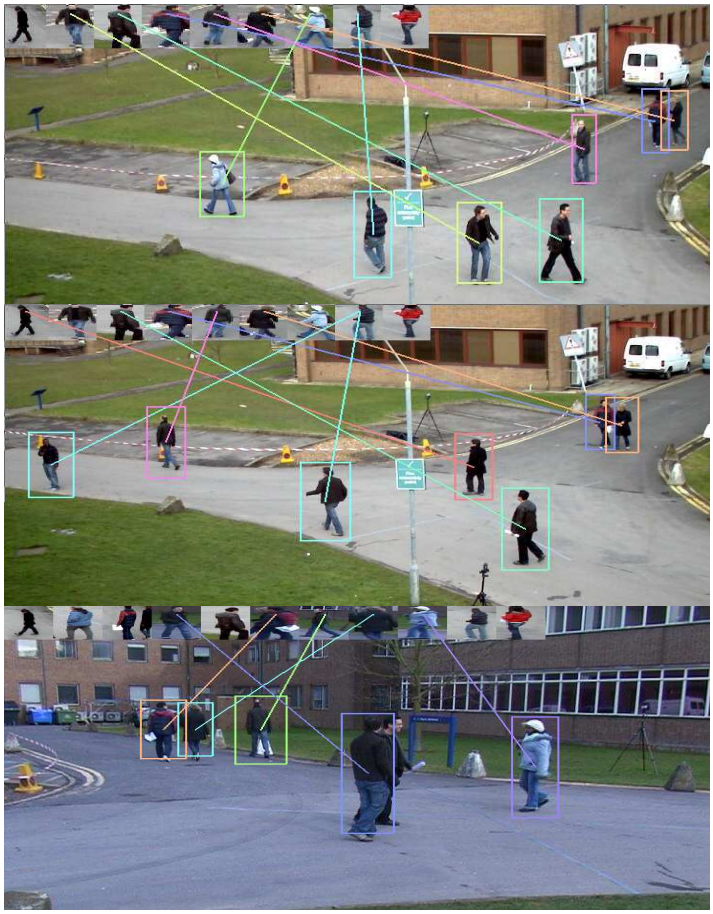


Figure 4-19: Some frames of PETS'09 showing the re-identification at different times and points of view.

To reduce the re-identification computational cost we propose a different arrangement of *MRC* matrices. Images are re-scaled to 96×128 pixels, then, rings of concentric rectangles are formed around the image center with exponentially increasing areas allowing some area overlapping. The proposed pattern is inspired in FREAK and DAISY [1, 132] but for rectangular covariance regions, these computations are performed fast thanks to the integral image method.

In our configuration, a total of 43 *MRC* descriptors were employed, **Table B.1** lists of the coordinates used to describe pedestrians (all samples have been resized to 128×96). Most of the rectangles are concentrated at the center and have a smaller size, more variability is tolerated at the periphery using bigger rectangles. To further simplify things *MRC* descriptors are compared one to one in contrast to [6] where comparison is maid sliding one grid against the other.

For the case *ELBCM*-based descriptors a different pattern of descriptors was used, the reason is that some regions are very small and the risk of not finding valid uniform LBP patterns when calculating the *ELBCM* covariance descriptors gets higher for these regions. To correct this problem, a different pattern of matrices which uses larger image regions was used, it is composed by the complete image (just using a small border to compute the LBP operators) and 4 quadrants. The list of five areas considered is given in **Table B.2**.

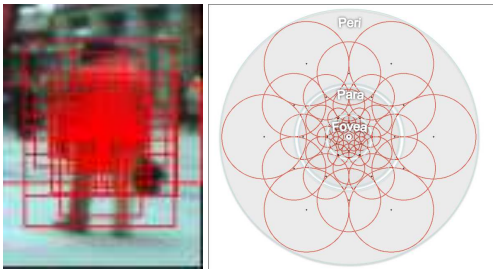


Figure 4-20: Proposed *MRC* pattern and its resemblance to FREAK

4.3.1 Target re-identification experiments

To validate our re-identification method the cumulative matching characteristic (**CMC**) of [6] and [58] is used. **CMC** represents the percentage of times the correct identity match is found on the top n matches. Tests were done on the **ETHZ** [121] and **PETS'09 L1-Walking-Sequence 1** [39] datasets.

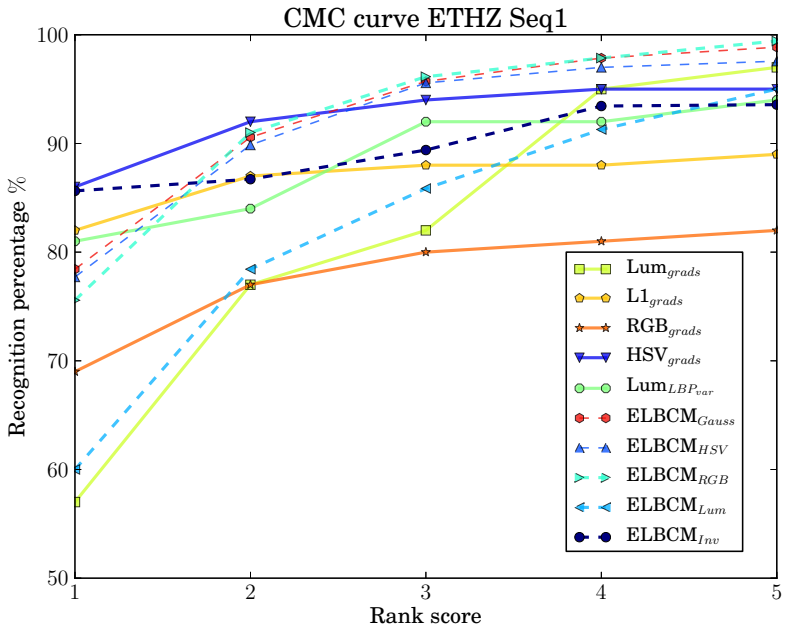
ETHZ dataset is composed by images from three different sequences (each one formed by 83, 35 and 28 individuals). Each individual, is captured by the same camera which suits just fine to our single-camera tracking objectives. For this experiment, eight different individuals from **PETS'09 L1-Walking-Sequence 1** were extracted taking discontinuous samples.

For each individual, 10 images were selected from the beginning ant the end and their **MRC** matrices (subsection 4.3.1) were calculated. The recognition rate was tested, taking random images and comparing against the registered signatures. Care was taken to avoid reusing any of the images occupied during signature calculation. Success is declared when the corresponding image identity is found inside the *top* n list.

To find out which feature combination is more discriminant, experiments were executed using the same feature combinations of the previous section listed in **Table 4.2**. The results of the experiments are shown in **Figures 4-21**, **4-22**, **4-23** and **4-24** where the cumulative matching characteristic for each dataset is reported. What these curves represent is the percentage of times the correct target class was ranked among the first K possible matches (*e.g.* rank score 1 measures the percentage of times the correct target class was ranked first).

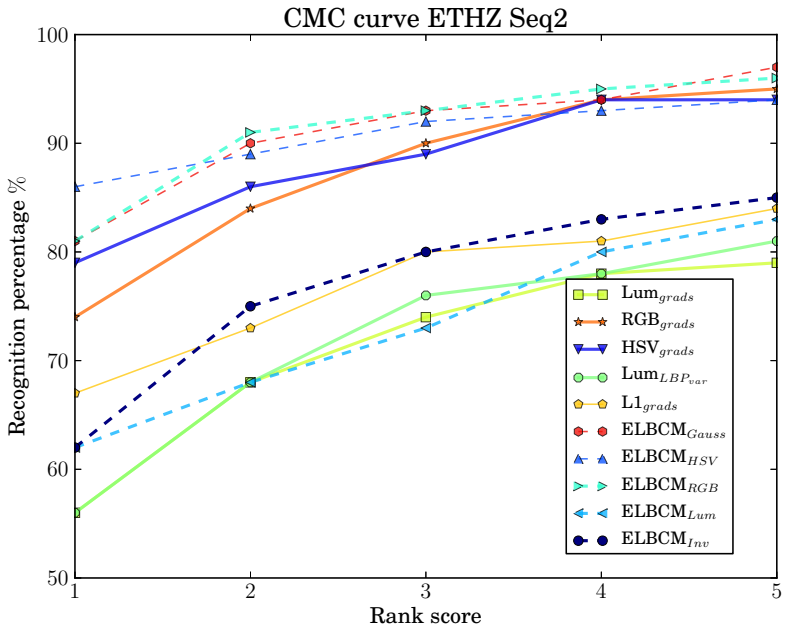
Features combinations based on the computation of directional gradients are denoted in continuous lines, dashed lines denote feature combinations based on the *ELBCM* operator. In most of the cases, features based on the *ELBCM* operator get a better score. Color information is very important to improve the discriminant capability of the descriptor, features based on luminance or in the color invariant get lower scores. However, in this experiments it is still not clear which color space is more discriminant, the most consistent one is probably the HSV color space that appear very well ranked for all the different dataset sequences.

The obtained re-identification rates are comparable to the ones reported in [6] employing a 75% less covariance matrices and fewer components inside them.



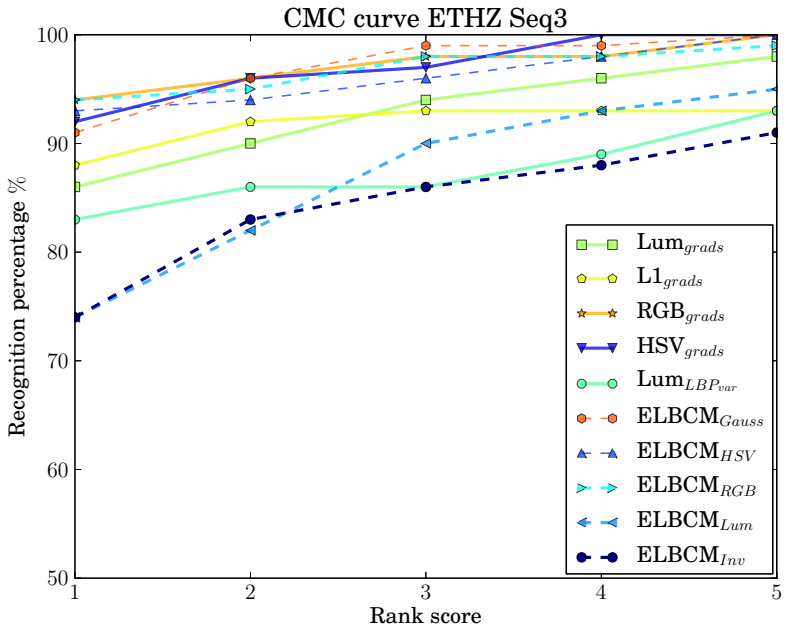
(a)

Figure 4-21: Cumulative Matching Characteristic for **ETHZ** sequence 1.



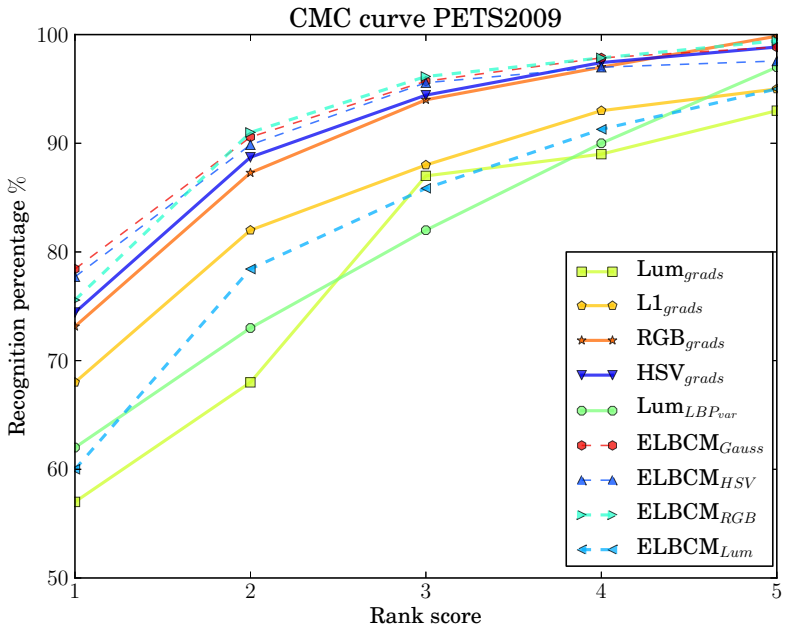
(a)

Figure 4-22: Cumulative Matching Characteristic for **ETHZ** sequence 2.



(a)

Figure 4-23: Cumulative Matching Characteristic for **ETHZ** sequence 3.



(a)

Figure 4-24: Cumulative Matching Characteristic for **PETS'09 S2-L1-View001**

4.4 Multiple object tracking

Multi-target tracking is considerably more challenging than single object tracking. Due to target inter-occlusions, appearance similarity between targets and other phenomena related to the presence of multiple targets in a scene, a lot more is required to solve this problem than just the mere application of single object tracking methods multiple times.

One of the great difficulties faced by multiple target tracking, is the state-space cardinality the algorithm is forced to deal with. The possible number of trajectories targets can follow is incredibly high (discrete case) or even infinite (continuous case). In addition, when objects suffer considerable changes in scale inside the tracking zone, the possible number of states grows even more. By introducing some dynamical constraints in the linear and angular velocity it is possible to reduce the number of possibilities into a physically plausible set trajectories, locations and scales.

The preponderant role that appearance information plays in the context of object recognition and tracking has been previously discussed. Information in the form of color or texture can be very useful to follow a target or to re-identify it after an occlusion. But taking advantage from this information is not always straightforward as many well known problems: occlusions, scale, illumination, appearance changes and background clutter contamination can drastically reduce the expected performance. Target representations need to be invariant and robust to all such phenomena, but unfortunately most color invariants, although robust against lighting changes, can lead to a reduction in the separability between targets increasing the number of matching ambiguities. To address the multiple object tracking problem a solution based on the *tracking-by-detection* paradigm is proposed.

4.4.1 Multiple object tracking algorithm description

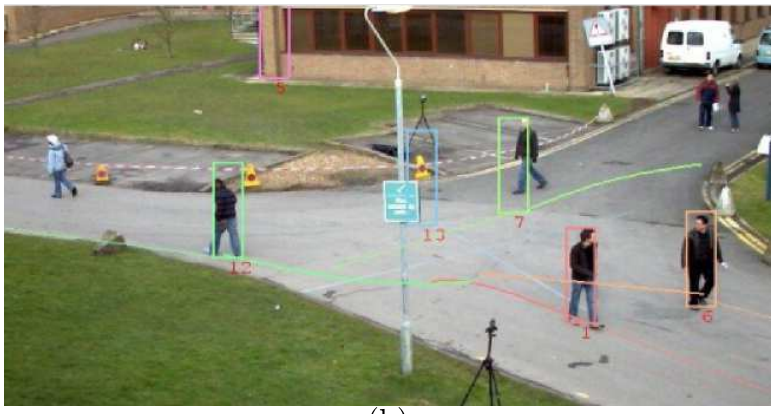
As a consequence of the continuous increase in the performance and speed offered by current object detection methods (HOG and classification in Riemannian manifolds discussed in **Section 2.1)**, multi-target tracking methods based in the *tracking by detection* strategy [2] are becoming very popular. Andriyenko and Schindler propose in [3] and [4] the use of discrete/continuous energy functions that evaluate the cost associated to a set of locations based on an overall energy function which considers target's dynamical properties such as linear and angular speed, trajectory persistence (preventing and penalizing unexpected target displacements) and target detections/disappearances occurring far from the tracking area borders.

4.4.1.1 Total Bregman Divergence (*TBD*) for appearance modelling

The possibility of including appearance information inside the energy function was left open in [3]. This idea is recovered here and enriched with the efficiency and robustness provided by covariance target representations.

As stated in **Section 1.2.5**, the set of covariance matrices lies in the Riemannian manifold formed by the set of symmetric positive definite matrices (*SPD*). A variety of metrics and divergences have been proposed in the literature to perform element by element comparisons on this set. Currently, the most popular metric for *SPD* matrices is the Riemannian metric of equation (2.30) (see details in B.0.6). Quite recently the Jensen-Bregman LogDet Divergence[29] has gained interest because it is cheaper to compute and because it enjoys some interesting robustness properties.

(a)



(b)

Figure 4-25: In every frame t our multi-target tracking receives a series of detections $\{\mathbf{d}_1 \cdots \mathbf{d}_n\}$ and consistently traces the object trajectories $\{\mathbf{p}_1 \cdots \mathbf{p}_m\}$.

The *Mean Riemannian Covariance* (*MRC*) discussed in **Section 4.3** blends together multiple observations of the same target into a general model. This *MRC* operator has many disadvantages: it is expensive to compute, there is no closed form definition (it is calculated iteratively) and it is affected by outliers. The geometric median [40] (**Section B.0.7.4**) is less sensible to outliers but it is expensive to compute and its existence is not completely guaranteed. More recent articles ([29] and [86]) propose other dissimilarity measure/divergences that alleviate these difficulties.

The total Bregman divergence (*TBD*) [86]

$$\delta_{TBD}(P,Q) = \frac{\log\left(\det\left(P^{-1}Q\right)\right) + tr(Q^{-1}P) - d}{2\sqrt{c + \frac{\left(\log\left(\det Q\right)\right)^2}{4} - \frac{d(1+\log 2\pi)}{2}\log\left(\det Q\right)}}, \tag{4.8}$$

where $c = \frac{3d}{4} + \frac{d^2 \log 2\pi}{2} + \frac{(d \log 2\pi)^2}{4}$ is a robust and efficient dissimilarity measure. The *TBD* and ℓ_1-norm based centre $\bar{Q}$ (known as t-centre) robustly represents a set of covariances $\{Q_i\}_{i=1}^n$ as

$$\bar{Q} = \left(\sum_{i=1}^{n} \frac{w_i^{-1}}{Q_i}\right)^{-1}, \tag{4.9}$$

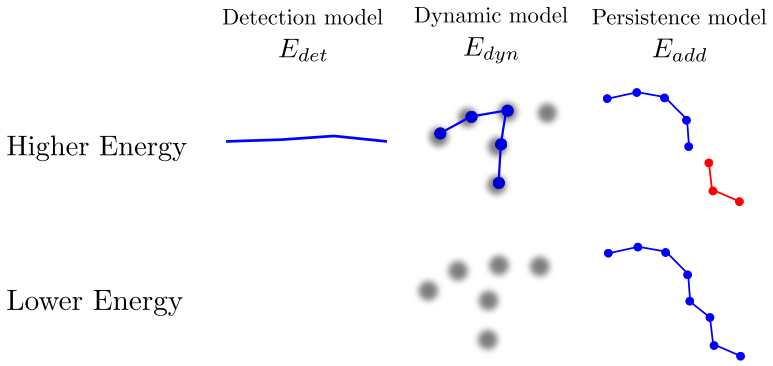


Figure 4-26: Decisions are made considering the addition of multiple energy terms. Top row shows the configurations that render higher energies for each individual term, lower energies are exemplified in the lower row. Detection peaks are denoted in gray.

where

$$w_i = \frac{\left(2\sqrt{c + \frac{(\log(\det Q_i))^2}{4} - \frac{d(1+\log 2\pi)}{2}\log(\det Q_i)}\right)^{-1}}{\sum_j \left(2\sqrt{c + \frac{(\log(\det Q_j))^2}{4} - \frac{d(1+\log 2\pi)}{2}\log(\det Q_j)}\right)^{-1}}. \tag{4.10}$$

Equations (4.8) and (4.10) may look complicate, but they are much more economical than Riemannian metric based alternatives because they have an analytic expression and only require the calculation of matrix multiplications and determinants (logarithms in these equations are *scalar* logarithms and not *matrix* logarithms), in contrast to the Riemannian-metric alternatives which are based on complicated calculations that require the computation of matrix eigenvalues among other expensive calculations.

4.4.1.2 Discrete energy modelling for multiple object tracking

The aim of our multi-tracking algorithm is to consistently detect, identify and trace object locations trough time. Ideally, the number of trajectories by object is exactly one, but inter-object occlusions, disappearances i.e., background occlusions, and false negatives in the detection method may cause trajectory drifts and fragmentations. At every time frame t the algorithm receives a set of detections $D_t = \{\mathbf{d}_1, \cdots, \mathbf{d}_m\}$ and the set of active paths (tracklets) $P_t = \{\mathbf{p}_1, \cdots, \mathbf{p}_n\}$. Each tracklet $\mathbf{p}_j$ (indexed by j), contains historical information about the object $\forall t \in \{t_{start}, \cdots, t_{end}\}$ where t_{start} is the frame where $\mathbf{p}_j$ was created and t_{end} the last frame where information about $\mathbf{p}_j$ is available. Tracklet $\mathbf{p}_j$'s location at t is denoted as $\mathbf{x}_j^t$ and its appearance description (covariance matrix) as $\mathbf{C}_j^t$. From these appearance descriptions the appearance model $\bar{\mathbf{C}}_j$ is calculated using the TBD t-center as in equation (4.10). A linear fitting function (i.e. least-squares) applied at time t, provides us the series of predicted locations $\hat{\mathbf{x}}_j^t, \forall t \in \{t_{end}+1, \cdots, t_{end}+N_f\}$ for every active path $\mathbf{p}_j \in P_t$ (N_f defines the length of the predictions).

Frame by frame the algorithm evaluates all possible matches between detections $\mathbf{d}_i$ and tracklets $\mathbf{p}_j$. New paths are created when it is required. At the end, the algorithm outputs a dynamic mapping set M_t containing the final set of tuples $M_t = \{(\mathbf{d}_i, \mathbf{p}_j)\}$ that minimize a discrete energy function. This idea is illustrated in Fig. 4-25-a where a set of points is marking locations of the detections (old detection are marked in a lighter red). In Fig.4-25-b these detections have been

grouped into trajectories (tracklets) tracing the evolution of object locations in the scene. The energy function $E(\mathbf{d}_i, \mathbf{p}_j)$ rewards plausible configurations and penalizes unreasonable ones. Our energy function includes most of the terms presented in [3]: a detection term based on the image detection data E_{det}, a physically motivated term which models the object dynamics (linear and angular velocities) E_{dyn}, an appearance term E_{app} that measures the similarity between the appearance of the target at the current location and compares it against tracklet's $\mathbf{p}_j$ covariance model. The last term considered by this energy functions penalizes path creations (or disappearances) favoring persistent and continuous trajectories E_{add}. All the aforementioned terms are accumulated as it is expressed in the following equation

$$E(\mathbf{d}_i, \mathbf{p}_j) = \underbrace{E_{det}}_{\text{detection confidence}} + \underbrace{E_{dyn}}_{\text{dynamic model}} + \underbrace{E_{app}}_{\text{appearance similarity}} + \underbrace{E_{add}}_{\text{persistence model}} . \tag{4.11}$$

Figure 4-26 shows how each one of this terms collaborates to detect the more plausible configurations.

Detection model: Objects are detected with a sliding window that measures a classifier response using **HOG** or classification in Riemannian manifolds (Sections 2.1.1.1 and 2.1.3.2). The energy term E_{det} uses the predictions $\hat{\mathbf{x}}_j^t$ inside $\mathbf{p}_j$. Energy is lower when $\hat{\mathbf{x}}_j^t$ passes trough a region of high pedestrian likelihood:

$$E_{det}(\hat{\mathbf{x}}_j^t) = \lambda + \sum_{g=1}^{D(t)} \frac{-c}{\|\hat{\mathbf{x}}_j^t - \mathbf{d}_g\|^2 + c} \tag{4.12}$$

In equation (4.12), $D(t)$ represents the number of detection peaks of frame t and $\mathbf{d}_g$ is the location of peak g. The value of λ penalizes predictions $\hat{\mathbf{x}}_j^t$ lacking of image evidence.

Dynamic Model:: The motion term E_{dyn} assumes a constant velocity model

$$E_{dyn} = \alpha \|\mathbf{v}_j^t - \hat{\mathbf{v}}_j^{t+1}\|, \tag{4.13}$$

where $\mathbf{v}_j^t = \dot{\mathbf{x}}_j^t = \mathbf{x}_j^t - \mathbf{x}_j^{t-1}$ is the current velocity vector of path p_j, and $\hat{\mathbf{v}}_j^{t+1}$ is estimated considering the detection location $\mathbf{d}_i$ as $\hat{\mathbf{v}}_j^{t+1} = \mathbf{d}_i - \mathbf{x}_j^t$.

Appearance model: Every detection $\mathbf{d}_i \in D_t$ has covariance descriptor $\hat{\mathbf{C}}_i$ which is compared against each $\mathbf{p}_j \in P_t$ through their associated t-center $\bar{\mathbf{C}}_j$. In the case of strong similarity the energy function is awarded with a negative value $-\gamma$, if not the energy functions grows in proportion to the *TBD* divergence $\beta\delta_{TBD}$ as

$$E_{app} = \begin{cases} \beta\delta_{TBD}(\hat{\mathbf{C}}_i, \bar{\mathbf{C}}_j) & \text{if } \delta_{TBD}(\hat{\mathbf{C}}_i, \mathbf{C}_j) > \delta_{max} \\ -\gamma & \text{if } x < \delta_{max}. \end{cases} \tag{4.14}$$

Persistence Model: When the algorithm find correspondences between detections and paths, it also considers the possibility of a new object entering the scene. This hypothesis is penalized in proportion to the Euclidean distance between the detection $\mathbf{d}_i \in D_t$ and the tracking area borders.

4.4.1.3 Energy minimization strategy

Our energy function is certainly nor linear neither convex, this problem is tackled evaluating every pair of combinations resulting from the crossing between the set of detections D_t with the tracklets set P_t. This way our problem reduces to a classical combinatorial optimization problem that can be efficiently solved by Munkres algorithm [99] (the Hungarian method) as in [72]. For each detection $\mathbf{d}_i \in D_t$ the minimization algorithm builds a list of candidate paths measuring the distance between $\mathbf{d}_i$ and the predicted location $\hat{\mathbf{x}}_j^t$ of the active path $\mathbf{p}_j \in P_t$. The *TBD* between the $\mathbf{d}_i$'s covariance descriptor $\hat{\mathbf{C}}_i$ and $\mathbf{p}_j$'s model $\mathbf{c}_j$ may also be considered. Paths whose predictors at frame t are outside a circle of radius r from the $\mathbf{d}_i$ and whose covariance model is extremely dissimilar are considered impossible combinations and an infinite energy is assigned accordingly. In the case of few or no paths close to any of detections, the *null* candidate is added indicating the creation of new path. Once the the algorithm has a complete list of candidates with their respective energies, it creates a new list of non-exclusive combinations $\{M_k\}_{k=0,\cdots,K-1}$. Munkres algorithm evaluates the cost of all these non-exclusive combinations and selects the less expensive one $M_t = \min(\{M_k\}_{k=0,\cdots,K-1})$. The complete processes is represented compactly in **Algorithm** 11.

Algorithm 11: Multi-target tracking by discrete combinatorial energy minimization.

Data: List of m detections D_t and the list of n paths P_t.

Result: Updated list of paths P_{t+1}.

1 For each detection $\mathbf{d}_i \in D_t$, $i = \{0, \cdots, m-1\}$ build a list of paths p_ℓ containing every $\mathbf{p}_j \in P_t$ whose predictors are located within a radius r from $\mathbf{d}_i$. If there are no paths close to $\mathbf{d}_i$ append c_0 to P_t to evaluate the cost of creating a new path.

2 From all the different p_ℓ's build the list of K possible candidate combinations: $\{M_k\}_{k=0,\cdots,K-1}$.

3 For each combination in $\{M_k\}_{k=0,\cdots,K-1}$, apply equation (4.11) to evaluate the energy associated to it.

4 Select the optimal combination $M_t = \min(\{M_k\}_{k=0,\cdots,K-1})$ having the minimal energy (using Munkres method).

5 Update the paths position, predictions and appearance model.

6 Create new paths as M_t indicates.

4.4.2 Multiple object tracking evaluation

The current best practice to evaluate multiple object tracking algorithms is to use of the CLEAR-metrics [127] which concretely defines what an ideal multiple object tracking algorithm should be able to do and proposes a set of measures to evaluate how good an algorithm is at this task. An explanation about these metrics is provided before passing to the analysis of the results obtained using the multiple tracking method proposed by this thesis.

CLEAR Multiple object tracking evaluation metrics

Following the definition proposed by the CLEAR metrics a multiple object tracking algorithm should be able to:

1. determine the correct number of objects present in the image at all points in time,
2. estimate the position of each object as precisely as possible,

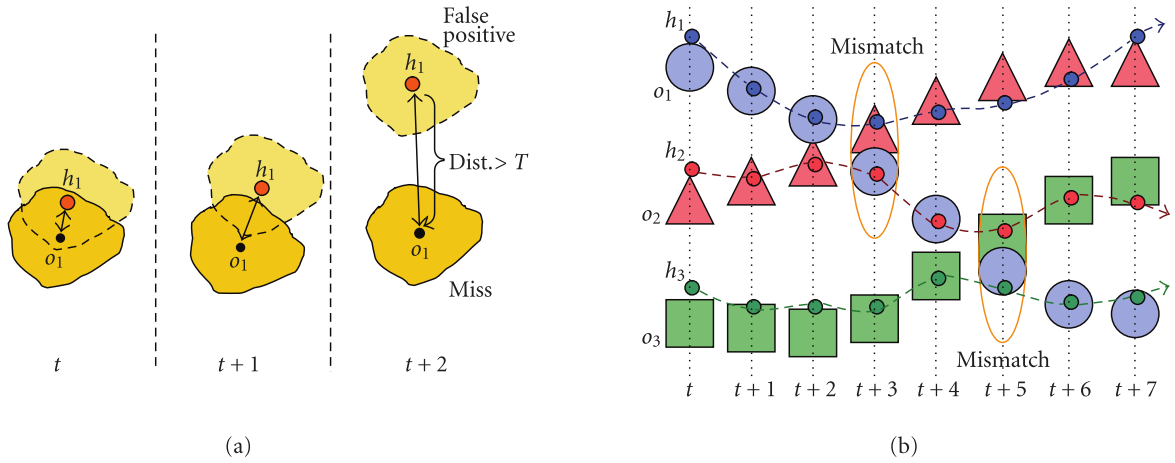


Figure 4-27: CLEAR matching error examples. In (a) the distance between o_1 and h_1 is computed, if it exceeds a certain threshold T, there is no valid correspondence o_1 is declared missing and h_1 becomes a false positive. Matching error examples are shown in (b), here h_2 is initially assigned to o_2, but after a few frames o_1 and o_2 cross paths and their trackers follow the wrong objects. Some frames later, o_2 swaps again with o_3. Image taken from [127]

3. keep a consistent track of each object over time,

4. assign a unique track identity and stay on it throughout the sequence.

The evaluation criteria proposed by CLEAR metrics is a set of mechanisms that enable the judgment of the precision of a tracker to determine the exact location of a target and reflect its ability to track consistently through time. They evaluate how good an algorithm is at tracing each one of the target trajectories and producing as few trajectories as possible per object.

For every frame t, a multiple object tracker outputs a set of hypothesis $\{h_1, \cdots, h_m\}$ for a set of visible objects $\{o_1, \cdots, o_n\}$. Different kinds of errors can occur at this point, Figure 4-27-(a) illustrates the precision error where the distance between the hypothesis and the actual target position is measured, if it exceeds a certain threshold T, the correspondence is invalidated (the object is declared missing and the hypothesis becomes a false positive). Examples of matching errors or identity swaps are given in Figure 4-27-(b).

At each time t the evaluation procedure follows these steps:

1. Establishes the best correspondence between h_j and o_i.

2. For each correspondence the error in the object's position estimation is computed.

3. Measures all correspondence errors which are accumulated as:

   (a) the total number of objects for which there is no output hypothesis (missed objects),

   (b) the total number of tracker hypothesis for which no real object exists (false positives),

   (c) the total number of identity changes or mismatch errors caused by swaps as two or more objects pass close to each other, or when an object track is reinitialized with a different track ID after an occlusion.

The performance of a multiple object tracking algorithm is thus expressed with two numbers: (i) the tracking *precision* which measures how well the exact positions of the objects are estimated, and (ii) the tracking *accuracy* which counts how

Table 4.3: Multiple object tracking evaluation metrics summary.

Term	Definition
$\{h_0, \cdots, h_t\}$	hypothesis set.
$\{o_0, \cdots, o_t\}$	visible objects set.
d_t^i	distance between the hypothesis h_i and the object o_i at frame t.
g_t	# of ground-truth objects at frame t.
c_t	# of matches at frame t.
m_t	# of false negatives (misses) at frame t.
fp_t	# of false positives at frame t.
mme_t	# of matching errors at frame t.
MT (Mostly tracked)	% of ground-truth trajectories covered for more than 80% of the sequence.
ML (Mostly lost)	% of ground-truth trajectories covered by a less than 20% of the sequence.
PT (Partially tracked)	$1.0 - MT - ML$.
MOTP	$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}$.
MOTA	$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t}$.
PF	Track fragmentations.
IDS	Identity swaps.

many mistakes the algorithm made in terms of false negatives (misses), false positives, mismatches and track recovery failures (swaps). Those metrics are proposed in a very intuitive way as:

1. The multiple object tracking *precision* (MOTP):

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}. \tag{4.15}$$

   which is the total error in estimating the position for object-hypothesis matches pairs over all frames, averaged by the total number of matches made (*i.e.* or the alignment error between the ground truth and the tracker output). This measure is independent of the skills of the algorithm to recognize object configurations and keep consistent trajectories,

2. The multiple object tracking *accuracy* (MOTA):

$$\text{MOTA} = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t} \tag{4.16}$$

   where g_t is the number of objects while m_t, fp_t and mme_t correspond to the number of misses, false positives and of mismatches, respectively, all of them for time t.

Other evaluation metrics used for testing were taken from [83]. The *mostly tracked* (MT) value represents the number (or percentage) of targets tracked correctly in more than 80% of the target appearance duration in the sequence, *mostly lost* (ML) targets are those tracked correctly for less than 20% of their duration in the sequence and *partially tracked* targets (PT) are those targets not considered by any of the other two categories. Table 4.3 summarizes all the evaluation metrics and terms discussed thus far.

Multiple object tracking evaluation results

To evaluate our approach we tested it four with widely used real world datasets as well as many other sequences of recorded in the laboratory. One of the tested sequences is **PETS 2009-S2L1-V1** from the **VS-PETS2009** benchmark[4] where only the first viewpoint is used. The other three datasets come from the **TUD** [5] dataset: **TUD-Campus**, **TUD-Crossings** and **TUD-Stadtmitte**. Ground truth is available for all these datasets. All the videos resulting from these experiments are available[6]. The experimental results are reported in **Table 4.4**, compared to the *state-of-the-art* results referenced in **Table 4.5**, they are not as accurate or precise as Milan *et al*. [95] or Berclaz *et al*. [12], but the advantage of the method proposed here resides on its ability to on run the fly, estimating the object trajectories frame by frame contrasting with [95] that requires the whole bunch of frames to estimate the optimal number paths and their most plausible trajectories.

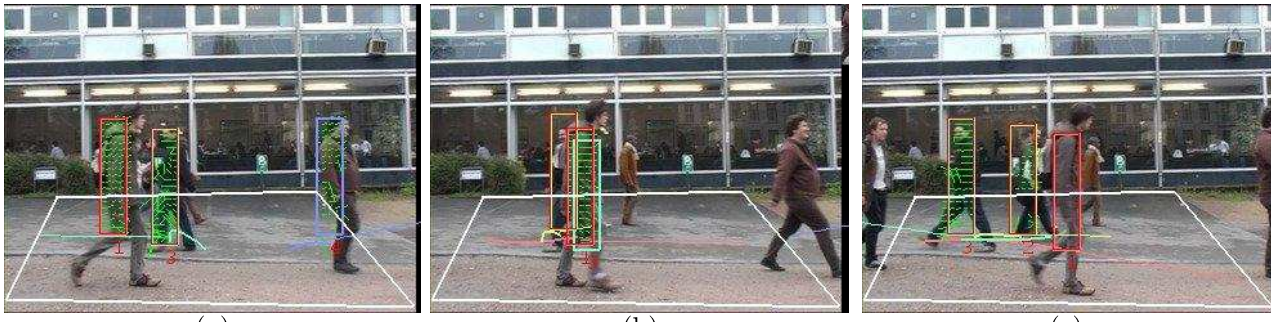
Table 4.4: Multiple object tracking evaluation results.

Sequence	MOTA (%)	MOTP (%)	MT	PT	ML	IDS
Pets 2009 S2-L1-V1	70.084	75.9	16	3	0	5
TUD Campus	74.92	63.4	5	2	0	0
TUD Satdtmitte	69.73	69	5	1	1	1
TUD Crossings	74.63	76.34	8	5	0	4

Table 4.5: Multiple object tracking *state-of-the-art* results, OM (Occlusion management plus appearance modeling) "Continuous Energy Minimization for Multi-Target Tracking" [95], KSP "k-shortest paths" [12], EKF "Extended Kalman Filter" [95], PF "Particle filter based tracking-by-detection" [20], HDA "Hierarchical data association" [64], CO "Continuous energy minimization" [3] and DCO "Discrete-Continuous Optimization" [4].

Sequence	Method	MOTA (%)	MOTP (%)	MT	ML	PF	IDS
Pets 2009 S2-L1-V1	OM+APP Milan *et al*. [95]	90.6	80.2	95.45	4.54	6	11
	KSP Berclaz *et al*. [12]	80.3	72.0	17	2	22	13
	EKF Milan *et al*. [95]	68.0	76.5	9	1	30	25
	PF Breitenstein *et al*. [20]	75	60	-	-	-	-
	HDA Hofmann *et al*. [64]	97.8	75.3	100	0	8	8
TUD-Stadtmitte	OM+APP Milan *et al*. [95]	71.1	65.5	77.7	0	3	4
	CO Andriyenko and Schindler [3]	60.5	65.8	100	0	4	7
	DCO Andriyenko *et al*. [4]	61.8	63.2	100	0	1	4
TUD Crossings	Yan *et al*. [150]	89.38	70.77	-	-	-	2
	Breitenstein *et al*. [20]	84.30	71.00	-	-	-	2
TUD Campus	Yan *et al*. [150]	84.82	67.76	-	-	-	0
	Breitenstein *et al*. [20]	73.30	67.00	-	-	-	2

[4] http://www.cvg.rdg.ac.uk/PETS2009
[5] http://www.mis.tu-darmstadt.de/node/428
[6] http://andresromeromier.wikispaces.com/



(a) (b) (c)

Figure 4-28: Target recovered correctly after occlusions in TUD-campus sequence.

4.5 Conclusions

This chapter presented the second and final group of contributions to the computer vision *state-of-the-art*. The *ELBCM* texture descriptor [115] is the first contribution, it proposes a more convenient way to embed LBP and texture information inside the covariance matrices than other descriptors based on a similar principle (*e.g. LBCM* [59] and GLRCD [154]). The experiments shown in **Section 4.1** improve the discrimination of the descriptor for different applications such as texture recognition, gesture recognition and tracking.

The influence of color inside the covariance descriptor was verified next using a variety of color representations. The set of classification experiments performed in **Section 4.2.1** shows how color can help to improve the discriminant power of the *ELBCM* descriptor. The robustness of the $L1$ color invariant [113] was verified too in **Section 4.2.2**. Based on both experiments it is possible to conclude that the $ELBCM_{Inv}$ and Inv_{grads} descriptors are as discriminative as the original $ELBCM_{Lum}$ descriptor (based purely on luminance) but less discriminative than other color descriptors based on RGB, HSV or Gaussian models. The utility of the L1-based descriptors resides on its resistance to brightness and saturation changes. RGB and HSV color spaces worked better than the Gaussian color models and their performance improves when combined with the texture information that *ELBCM* provides. In both experiments $ELBCM_{RGB}$ performed slightly better than the other feature combinations (except on *David* sequence where brightness and saturation changes occur).

Section 4.3 presented an object re-identification method [114] which is appropriate to assist multiple object tracking applications specially to handing off targets captured on different cameras (*i.e.* multi-view configurations). The discriminant power of the covariance descriptor was increased by calculating multiple covariance descriptors (*i.e.* forming an array or pattern) for each sample and averaging this set of covariance matrices for all the samples belonging to the same target.

The multiple object tracking algorithm presented in **Section 4.4** represents the last contribution of this thesis related to the computer vision field. The method presented here is not as accurate nor precise as the state-of-the-art methods proposed by Milan *et al*. but is capable of executing *on-the-fly* and in real-time.

In addition to the computer vision contributions presented up to this point, this thesis proposes a series of software optimizations and algorithm transformations to accelerate the execution of the covariance matching algorithm and adapt it to different computer architectures all theses aspects are treated in the next chapter.

(a) (b) (c) (d) (e) (f)

Figure 4-29: Frames taken from the **PETS2009-S2-L1-Time_12-34-View_001** sequence showing targets crossing each other and being occluded. The discrete energy minimization algorithm based on the togal Bremgan divergence is able to trace their trajectories and handle their identities.

CHAPTER 5

Real-time implementation: software optimizations and algorithm transformations

Contents

Introduction

Theoretical advances in mathematics, machine learning and signal and image processing domains together with improvements in electronics and computer architecture technology such as the ever increasing computing power attributed Moore's law (that states that the number of transistors on integrated circuits is doubled approximately every two years), or by trend imposed by parallel architectures are fueling the development of mobile technology applications in different domains such as automotive, health, telecommunications and many others. These applications involve a large diversity of algorithms *e.g.* 2D and multi-spectral filtering, pattern recognition, segmentation and classification, cryptography and communications.

The development process of all these algorithms is often divided into three sequential steps: the theoretical study of the algorithms, the study of the target architecture and *last but not least* the implementation.

Algorithms are as much a technology as every other aspect that makes computers work. In many cases, choosing the right algorithm is more relevant than being able to buy the latest and fastest hardware, or as Cormen puts it in [31]: "Even with the impressive advances that we continually see in computer hardware, total system performance depends on choosing efficient algorithms as much as on choosing fast hardware or efficient operating systems. Just as rapid advances are being made in other computer technologies, they are being made in algorithms as well.". In the same sense, the speed difference between a *poorly* implemented algorithm algorithm and a nice hardware implementation that considers all the relevant aspects and details about a particular target architecture can be of many orders of magnitude. Algorithm-architecture adaptations holistically observe the interactions between algorithm analysis aspects (number of arithmetical operations, memory loads and stores, etc) together with the targeted platform execution characteristics (instruction timings, memory capacities, buses speeds) and other particularities related to the chosen architecture such as vectorized instruction sets and multi-threaded execution to obtain sufficient levels of performance. In other words, application developers and software system architects need not only to select the appropriate algorithms to use, but also the means to implement them on hardware eliminating unnecessary operations and stalls and intelligently distributing the computing load between the different sub-elements of the architecture to minimize runtime overhead.

The core algorithm for modeling the appearance of the targets in this thesis is the covariance descriptor, the objective of this chapter is to describe the most crucial challenges and particularities that appeared when implementing and optimizing this algorithm on a variety of desktop processors (*e.g.*, 4-core Intel's Penryn, 8-core Nehalem and 4-core SandyBridge architectures) and on low-power processors suitable for embedded platforms such as Intel's ULV U9300 and ARM Cortex-A9.

This chapter is subdivided in two sections, **Section 5.1** opens with a short review of basic computer architecture concepts: the memory hierarchy, memory timings and hardware characteristics, the different levels of parallelism that exist, Flynn's taxonomy and Amdahl's law, the concepts of temporal and spatial data locality and some of the properties that a cache conscious algorithm should meet. Readers familiar with all these concepts can go directly to the second part of the chapter, (**Section 5.2**) which is dedicated to the analysis and optimization of the computation of the covariance matrix descriptor for parallel architectures. The integral images method used to accelerate the computation of the covariance descriptor (in order to scale the algorithm to an unlimited quantity of descriptors computed on a single image), is evaluated first (Section 5.2.2) to determine the numerical accuracy constraints that we must consider when implementing the covariance computation algorithm. After the numerical correctness of the algorithm is asserted, the baseline implementation of the algorithm is described and benchmarked to determine the most critical bottlenecks for this implementation. Then, the rest of the chapter describes the code and data layout adaptations necessary to vectorize and optimize the algorithm execution.

5.1 Short review of parallel computer architectures concepts

Targeted architectures description

This thesis formed part of the ITEA/SPY European project. The aim of this project was to create a new automated, intelligent surveillance and rescue framework adapted for mobile environments. Most of the algorithms developed for this thesis were designed for portable smart camera applications, this constraint pushed the participants to select an embedded multi-core general purpose processor (*GPP*) member of the ARM Cortex-A series. Because it is very suitable for embedded targets, C programming language was preferred over other high-level languages (C allows to interact so directly with the hardware that it is sometimes classified as a low-level programming language).

The development cycle was the following: algorithms were first implemented and tested on desktop computers (running a Linux environment), once the desired algorithms were validated they were ported to the selected embedded architectures (Intel U9300 and ARM Cortex A9) care was taken to avoid the use of libraries not existing on all the platforms. By respecting this practices, the resulting code is extremely portable, code developed for *GPP*'s remains very easy to design and maintain in contrast to other architectures such as graphic processing units (*GPU*'s), digital signal processors (*DSP*'s) and programmable logic electronics such as programmable logic devices (*PLD*'s) or field-programmable gate arrays (*FPGA*'s).

Intel+SSE and ARM+Neon

There are different ways (an levels) to represent the organization of a computer. The instruction set architecture (*ISA*) is the model which is closest to the programmer, it enlists the native data types, the instructions, the registers, the addressing modes, the memory architecture, the set of interruptions, how exceptions are handled, the buses to input and output (I/O) peripherals, etc. The micro-architecture (μ-arch) details how the instruction set architecture (*ISA*) is implemented on a processor, different architectures can implement the same *ISA* using significantly different semiconductor technologies and different interconnections between the operational elements, variations can go from single gates and registers to complete arithmetic logic units (*ALU*'s) and even more complex or larger elements.

It is not necessary to be a computer architect to develop good quality and efficient code. From the point of view of the programmer, it is only required to have a simple (but clear) mental representation of the organization of the hardware resources being used, most *GPP* architectures share the concepts of cores, arithmetical and logical units, vector units and cache memories. **Figure 5-1a** shows an sketch of a typical multi-core processor, every core contains multiple functional units each one designed to perform some type of arithmetic operations, cache memories are used to manage the trade-off that exists between memory speed and capacity.

Each processor has a central clock which controls the pace at which the semiconductors (*e.g.* gates, functional units) pass from one state to the next. Depending on the properties of the electronics components (*i.e.* the transistors that form the processor) the maximum allowed operational frequency of the processor clock changes. The frequency of a processor is important as it determines the duration of the instruction cycle. For so many years, the exponential increase on the frequency of a CPU resulted in a *free* speedup of numeric software. From 1973 to 2003 microprocessor clock rates increased by three orders of magnitude (from 1 MHz to 1 GHz). Legacy code in some sense benefited automatically of the improvements of speed of computer architectures. However, in the last years the perception is that this trend is languishing and that the

era *free-speedup* for legacy code is coming to an end [26]. Since 2005 CPU frequencies have stalled around 2 to 3 GHz and many factors are limiting the growth of achievable performance of single cores: the first important factor is related to the observed non-linear growth of power consumption as the clock rate increases which at the level computers are now turns out to be completely unacceptable (at 130W air cooling systems are no longer practical), clock rates stopped climbing because the power growth needed to be arrested and because of the emergence of mobile and embedded computing where the need for low-power consumption is much more important than it is for desktops and servers. A second problem is posed by the famous *discrepancy* that exists between the speed of the processor and memories. This problem renders the increase of the clock rates fruitless because many applications are fundamentally bounded by the memory performance and not by computing (i.e. loading data from off-line chips could be more expensive than performing many arithmetic instructions). To avoid this problem, algorithms need to be coded to avoid memory accesses (loads and stores) as much as possible.

Figures 5-2a and **5-2b** show similar sketches for the Intel Nehalem and ARM Cortex A9. Both processors are able to execute almost the same type of arithmetical operations. But each processor design involves a very complex set of technologies (implemented as circuitry) that vary significantly the inner functioning of a processor: mechanisms that vary the operational frequency of the processor to economize in energy depending on the load, to pre-fetch data and/or instructions, fetch and branch prediction mechanisms, to change the order of execution of the stream of instructions received depending on the data availability and their dependencies, etc. For the case of the Nehalem, **Figure 5-2a** shows the communication link to the main memory (*e.g.* DDR3 SDRAM) and to other peripheral using the Direct Media Interface (*DMI*) or QuickPath Interconnect (*QPI*) technology to implement what it is known as the *point-to-point* link that connects to the Platform Controller Hub (*PCH*). This circuit, handles the communication to different ports such as serial ATA (to communicate with the hard-drive), PCI Express, USB 2.0 ports, the network interface, etc. This last aspect is illustrated in **Figure 5-1b**. Most Nehalem desktop and mobile processors (e.g. Core i3, Core i5, and Core i7 processors) use *DMI*, but some Core i7 models and other non-consumer processors targeted for workstations and servers such as the Xeon Phi coprocessor[1] use a newer technology from Intel known as QuickPath Interconnect (*QPI*) point-to-point link that similarly to *DMI* handles the interconnection between the processor and the Platform Controller Hub (*PCH*). In more complex instances, separated *QPI* links are used to connect one or more processors and one or more *PCH* hubs to form a network on the motherboard, this technology allows all its components to access each other and enable a non-uniform memory access architecture (*NUMA*). *DMI* was originally designed to support 10 GB/s in each direction, *QPI* supports up to 25.6 GP/s.

The Intel ULV 9300 processor (from ultra-low voltage) belongs to the Penryn family which includes an extensive array of micro-architecture elements that boost performance across a broad range of software, this includes the Intel Streaming SIMD Extensions 4 (SSE4) that deliver further performance gains for SIMD (single instruction, multiple data) software (32-bit or 64-bit), the group of applications benefiting from this technology are: graphics, video encoding and processing, 3-D imaging, and gaming, image processing, etc. On the other side, the Cortex-A9 processor offers a good balance of computing power and energy consumption, its floating-point arithmetic unit executes single and double precision scalar operations two times faster than the preceding ARM generation, its Neon Media Processing Engine allows the programmer to accelerate media an signal processing functions using the ARM Neon Advanced SIMD instruction set which supports the execution of rich SIMD operations over 8, 16 and 32 bit floating-point data quantities every cycle.

[1] Intel Xeon Phi coprocessors provide up to 61 cores, 244 threads, and 1.2 teraFLOPS of performance.

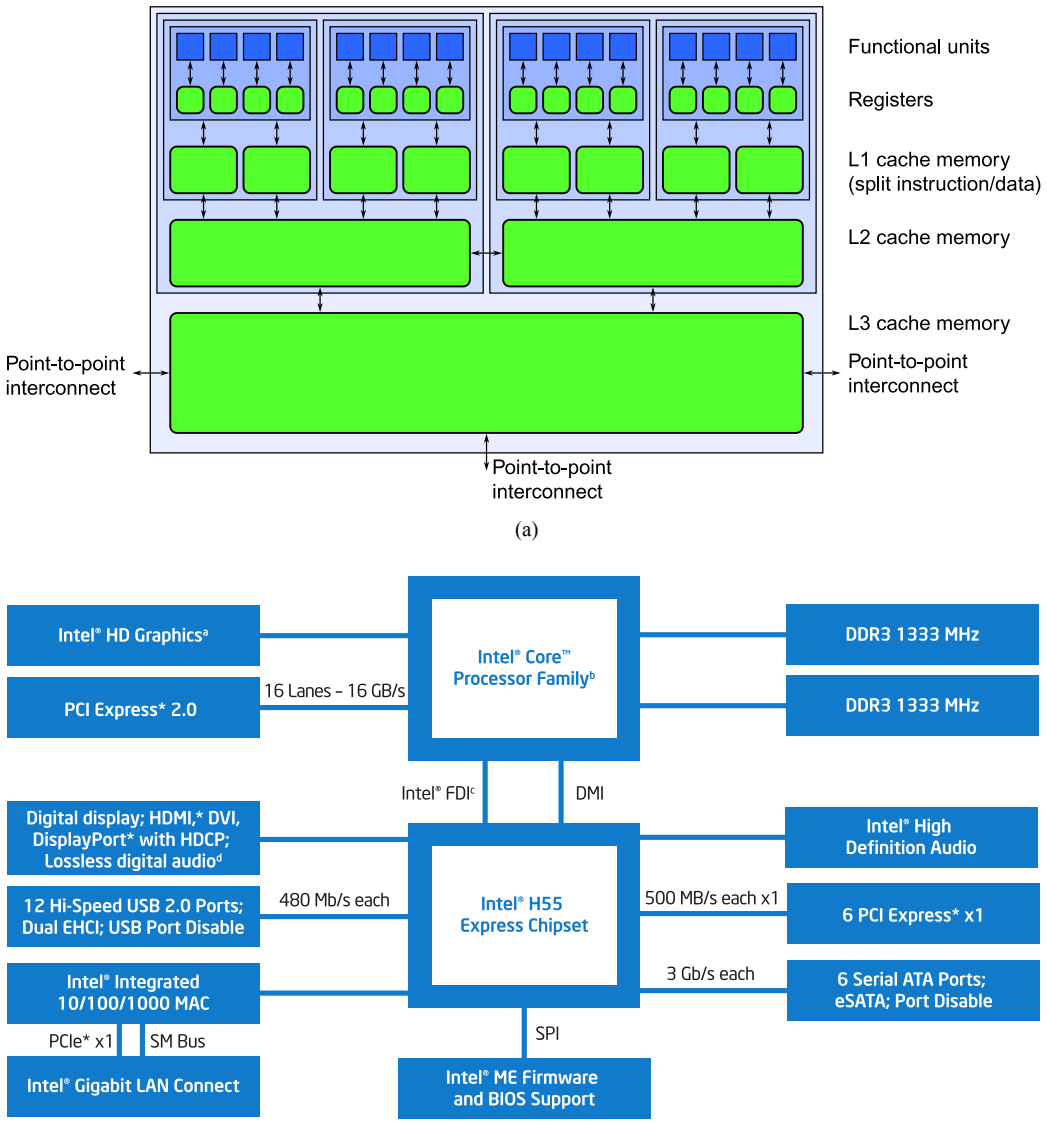


Figure 5-1: (a) A multi-core processor with hierarchical cache. Each core has its own registers and functional units, typically instruction and data caches are separated in the highest level. Larger and slower caches are shared with other cores forming a hierarchical structure of memory. (b) An schema showing how Intel Core processors are connected to the main memory (DDR3 SDRAM blocks) and using the *DMI* point-to-point interconnect to access other peripheral. Figures taken from [93] and from Intel H57 and H55 documentation. [2]

[2] http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/h57-and-h55-express-chipset-brief.pdf

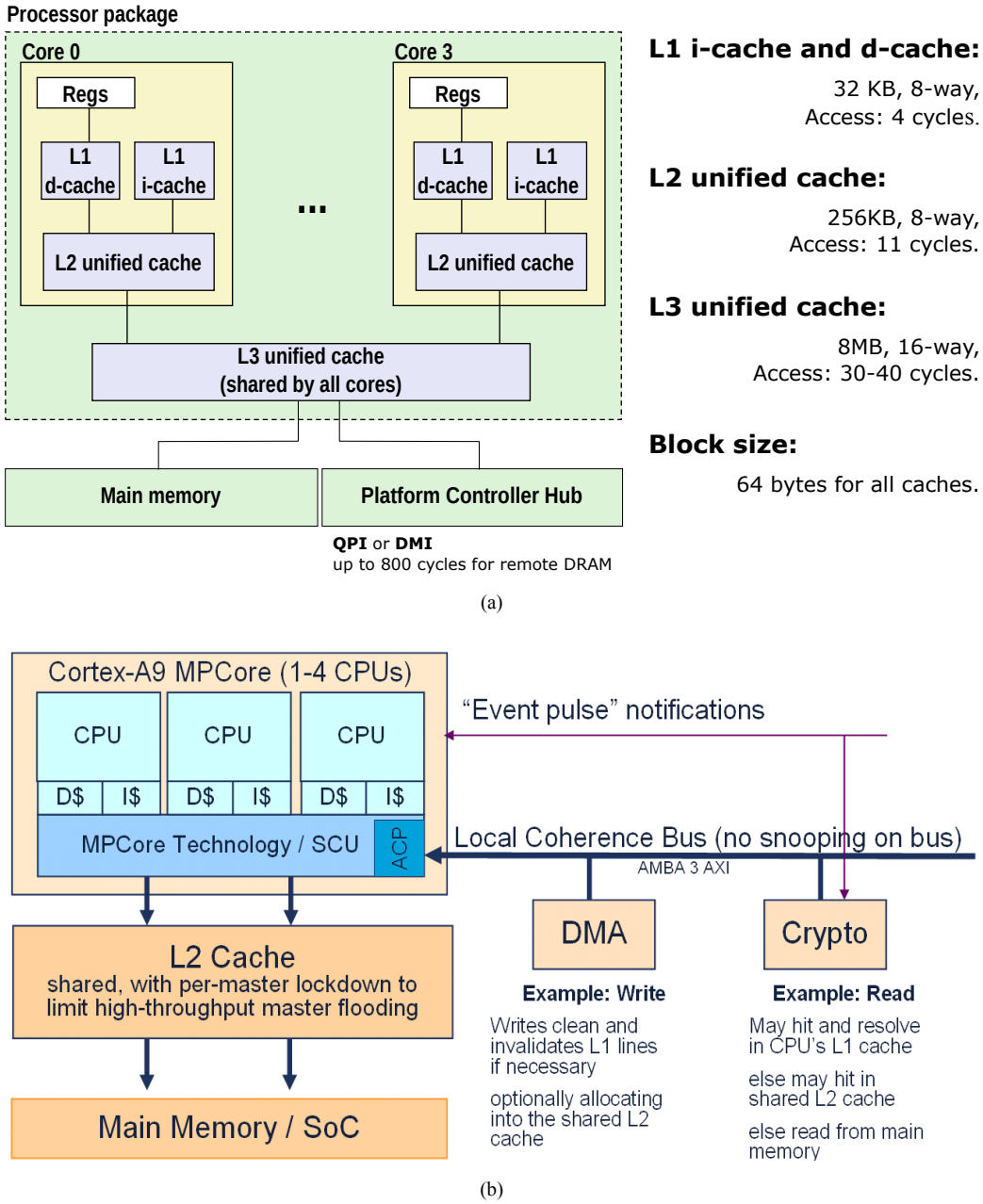


Figure 5-2: Targeted architectures

5.1.1 Basic computer architecture concepts

Registers and cache memories

Due to the enormous disparity that exists between the speed of the processor and the memories, computer architectures are commonly structured in a hierarchical way. Their structure is given by processor registers, multiple cache levels, main memory, network, hard-drives and other peripherals. **Figure 5-1a** shows a sketch or abstract representation of a typical **multicore processor**. Inside every core we have the functional units, each one is charged of performing a particular arithmetic operation (they are the basic unit of computation). Registers are the units of memory which are closer to the functional units, they can be used to feed the functional units with operands. CPUs have very few registers and each one offers very few bytes of space. Caches are next type of memories in the hierarchy, they are slightly slower than the core registers but they offer more space. Caches are separated in multiple levels, each level being slower but larger than the one above it. In multi-core systems, only the lowest cache levels are shared between the cores, caches are organized by storage blocks called cache lines. A cache line is typically much larger than the processor's word length and usually larger than a vector register. Common cache line sizes are between 64 bytes and 128 bytes. SIMD registers in SSE are 128-bit wide that is 16 bytes wide, a cache line of 64 bytes length (such as Nehalem) is 4 times longer than the SSE vector register. Cache lines are used to populate the cache memory every time a data is read from memory, the objective of transferring the entire line instead of just data by data is to amortize the overhead of setting up the transfer. Modern day CPUs include circuitry that analyzes the memory access patterns and prefetches data and instructions to the cache, but for being effective, memory accesses should not jump around indiscriminately in memory so that the extra data read into the cache is fully exploited. Peak memory access performance is obtained only when memory is accessed following a predictable sequence of addresses, so that the program makes full use of the line transfers and prefetching data mechanism are efficient.

Caches are much smaller than main memory, they use **SRAM** (static random access memory) and they are significantly faster than other types of memory used for main memory such as **DRAM** (dynamic random access memory). The most important limitations of SRAM is that they have a much smaller capacity per unit area and that they are more costly, large on-chip cache memories currently are on the order of 8 to 20 MB. So, cache memories cannot contain all the code and data of the executing program at a given time, so there is a lookup table in the cache that is used to respond to a memory reference, a separate D-cache is used for fetching data and a I-cache to fetch instructions, if the memory reference is mapped in the cache, then we have a **cache hit** and the instruction or data can be sent right away to the processor, otherwise we have a **cache miss** and the next levels are recursively probed until the missing item is found. Cache misses are an important source of delays that usually take between 20 to 100 clock cycles, this is comparable to the magnitude of time required to perform more than 100 floating point operations, ideally, a large number of on-chip compute operations should be performed for every off-chip memory access, thus a good indicator of the performance of an algorithm is obtained by measuring the ratio of computation *vs.* the number of memory accesses, this ratio is commonly called the **arithmetic intensity**.

The **hit time** expresses the time it takes for a cache memory to bring a line in the cache memory to the processor, this time depends on the hierarchy of the cache and for the case of multi-core architectures if the cache memory is shared or not with the other cores. **Table 5.1** briefly describes the required time to access a data line depending on the hierarchy level of the memory for the Intel Nehalem processor [36], according to it, the additional time required when a cache miss occurs

(depending on its level) can go from 60 ns up to 100 ns. For a remote DRAM access (data that is resident in the DRAM on a remote socket) the number of cycles can go up to 800 cycles instead of just 4 that L1 cache hit takes. The percentage of the memory accesses for instructions or data which are not found in the cache memories is often referred as the algorithm's *miss rate* (expressed as # misses/# accesses). For the L1 cache memory typical miss rate values range between 3% and 10% depending on the algorithm characteristics, the cache memories size and the memory layout. Similarly, the L2 cache miss rate values should be $< 1\%$. If the greatest part of the memory accesses are covered by the cache memory (cached memory locations) the average latency of the memory accesses will be similar to the cache latency instead of the main memory latency.

Table 5.1: Description of the required time to access a data line depending on the hierarchy level of the memory for the Intel Nehalem processor [36].

Memory access	Latency (approximately)
L1 cache hit	4 cycles
L2 cache hit	10 cycles
L3 cache hit (line unshared)	40 cycles
L3 cache hit (shared line in another core)	65 cycles
L3 cache hit (modified in another core)	75 cycles
Remote L3 cache	$100 - 300$ cycles
Local DRAM	60 ns
Remote DRAM	100 ns

How data is stored in memory has a clear impact on the execution speed of an algorithm, this is due to the *principle of locality* which analyzes how a value or a set of related storage locations is accessed. What locality measures is how predictable an algorithm or system behaves, algorithms exhibiting strong locality are highly appreciated because they are strong candidates for performance optimization. There are two basic types of reference localities: *temporal locality* which refers to how often a specific data, address or any other type of resources is accessed within a certain period of time, and secondly *spatial locality* which refers to the use of elements stored in nearby locations in the near feature. A good locality means that the hardware can win its bet since the supposed prediction conditions are more likely to be true reducing the required communication. Each memory access pulls out an entire cache line around that memory location onto the chip and into the cache memories, using the contents of that line repeatedly while it is still residing in the cache memory is faster than pulling those contents from base memory multiple times. Suboptimal data layouts in memory are a common reason for weak data locality and consequently of poor performances. Thankfully, some simple transformations such as: loop re-arrangements and data layout changes may help to improve the spatial locality, in all cases it is worth to keep in mind that communication is very expensive and computation very cheap that sometimes it is better to increase the required amount of work in exchange of reducing communication.

Instruction sequencing is improved using some of the following code adaptations: loop interchange, loop fusion, loop unrolling and loop unwinding. Examples of these transformations are provided in the **Appendix C** for consulting. A different nature alternative is to use a technique called **buffering**, which consists in copying data into contiguous temporary buffers to overcome cache misses due to cache associativity, when working on multi-dimensional data structures such as matrices and images, logically close elements can be far from each other when mapped to a linearized memory. Cache

associativity and cache line size are in conflict if the programmer wants to hold, for instance, a small rectangular section of an image or a matrix in cache. One simple solution is to copy the desired block into a contiguous temporary buffer, this may have an initially high cost but alleviates cache thrashing if the data section is accessed frequently.

Main memory

Main memory is the next element in the memory hierarchy of a computer architecture. It is is usually based on **DRAM** (dynamic random access memory). Because of its structural simplicity: only one transistor and a capacitor are required per bit, compared to four or six transistors that static (*SRAM*) requires, so *DRAM* offers a higher memory density but the capacitors charge needs to refreshed periodically. A typical DRAM chip of today is based on synchronous dynamic random access memory or **SDRAM** which is just a type of DRAM with a synchronous interface to give response to the clock signals sent by the computer's system bus. Depending on the model, main memories offer a different **bandwidth** and **latencies**. Bandwidth refers to the rate at which data can be read from or stored into a semiconductor memory by a processor, it is commonly expressed in terms of bytes per second. Latencies refer to the number of cycles or timing delays (usually in nano seconds) that occur when transmitting data between the CPU and the memory semiconductor. The most widespread implementation of SDRAM technology in current day computers is DDR3 (double data rate v3 SDRAM), depending on the model and the manufacturer bandwidth and latencies change. For instance, for the case of a DDR3-1333 chip with a latency of 7 (CAS latency) the wait time for a transmission is 10.5 ns, bandwidth is calculated by taking the number of transfers per second and multiplying by the number of bytes that fit on the data bus (for a 64-bit data bus this factor is eight). For a processor operating at 2 GHz (cycle time of 0.5 ns) transferring data to/from a DDR3-1333 chip would approximately require 21 cycles.

5.1.2 Basic parallel architectures concepts

Levels of parallelism

Parallelism comes in different levels and forms:

- **Single thread instruction level parallelism (ILP)**: hardware and microprocessors include multiple mechanism (implemented in circuitry) to extract parallelism from serial instruction streams. If a group of nearby instructions have no dependencies between them, modern processors can re-schedule (**out of order execution**) or execute them in parallel using different arithmetical or logical units. This technique is often referred as **super scalar processing**, processors are equipped with multiple redundant units that enable the execution of more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to the processor. Super scalar processing is different than multi-core processing in the sense that each functional unit is not properly a complete CPU core but only an additional execution resource within a single CPU (*e.g.* arithmetic unit, a bit shifter, a multiplier). Very Large Instruction Word (**VLIW**) processors (commonly used by Texas Instruments DSP families) use a related technique, the scheduling of the instructions is analyzed in advance by an specialized compiler or manually by the programmer.

- **Data level parallelism**: Data parallelism consists in distributing the data across different computing nodes and computing the same operations on multiple datum simultaneously, there should not exist any dependency between the input data and the results obtained for other data elements. Data parallelism allows to modify multiple values simultaneously using the same modification function. Some implementations of this type of processing are obtained by vector processing using SIMD instructions. Data parallelism is more oriented around data rather than the execution of multiple tasks concurrently, it is closely related to the width of the hardware registers and the number of data elements that can be processed simultaneously following a single control flow. In data parallelism the same flow of control is applied on multiple data elements.

- **Thread level parallelism (TLP)**: This type of parallelism occurs when a program is separated into independent tasks and each task can be performed in parallel without dependencies between each other. In single threaded cores each thread is executed in a separate core and each one keeps its own state (*e.g.* program counter, state registers) necessary to execute independently their own stream of instructions. Using multiple instruction streams improves the execution times of multi-threaded programs.

For long, programming was done by modeling computers as if they were serial machines, meanwhile, computer architects and compiler designers worked hard to find ways to automatically extract ILP parallelism, this helped to keep things simple for the programmers and allowed them to ignore the true parallelism that existed in hardware but most computer architects believe that these techniques have reached a limit. New processor generations need to avoid their dependency on the rise of clock rates or on automatic IPL mechanisms.

Flynn's taxonomy

The way processors combine control flow and data management define the type of parallelism available on different architectures, Flynn presented in [41] a classic categorization of processor architectures depending on whether they have multiple streams of data, multiple flows of control or both.

- **Single Instruction, Single Data (SISD)**: The standard and typical non-parallel uniprocessor.

- **Single Instruction, Multiple Data streams (SIMD)**: A single operation is executed simultaneously on multiple elements of data by broadcasting the operations on multiple data paths. SIMD processors are also known as array processors.

- **Multiple Instruction, Single Data (MISD)**: no such machine exists.

- **Multiple Instruction, Multiple Data (MIMD)**: This processors execute separate instruction streams, each with its own flow of control and operating on different data. This model characterizes the use of multiple cores integrated in a single processor, or multiple processors in a single computer and multiple computers forming a cluster. When different processors of different architectures are present in the same computer, we have a heterogeneous computer.

Vector processors appeared in the 1970's and were the basis of most supercomputers through the 1980's and the 1990's. Thanks to the growth of the registers data width today's most commodity CPU's implement architectures featuring vector

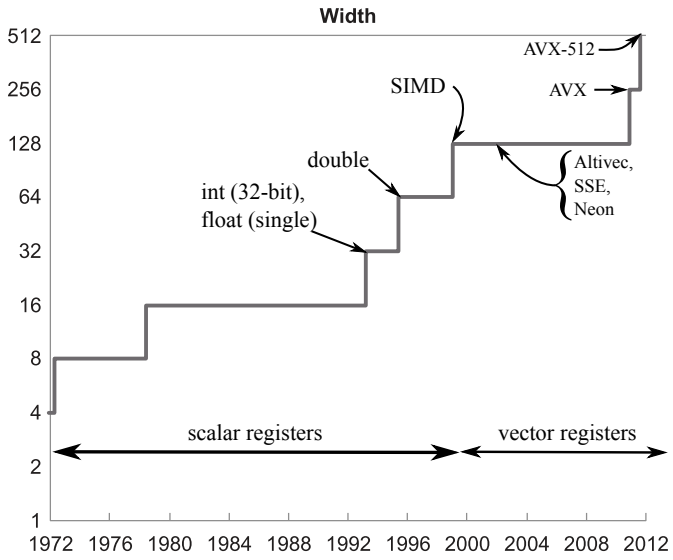


Figure 5-3: Growth in data processing widths (log scale) measured as the number of bits in registers over time. Vector (SIMD) instructions specify the processing of multiple scalar elements at once. Image taken from [93].

instructions. A measure of the growth of data width parallelism is shown in **Figure** 5-3, as years passed by integer arithmetic evolved increasing the number of bits (from 4 to 32 bit), in the 1990's floating point units for single (32-bit) and double precision (64-bit) where integrated to Intel's and AMD's CPUs and it is until 1997 that 128-bit vector registers were introduced. The first processor family from Intel equipped with this technology where commercialized with the name of Pentium MMX, they were limited to perform integer arithmetic instructions only. SIMD extensions able to process single precision floating point vectors where introduced in 1998 by: AMD (3DNow!), Intel (SSE) and Motorola (AltiVec). AltiVec was the first vector instruction set able to handle both integer and floating-point vectors, variants of this technology which is designed and own by Apple, IBM and Freescale were implemented too on the PowerPC architectures and other related processors such as the Cell Broadband Engine used by the PlayStation 3. This trend of evolution continues to the present day, Intel introduced this year (2013) its AVX-512 instruction set (included by the Xeon Phi coprocessor) which allows to perform vector integer and single or double precision floating-point operations in 512-bit wide registers, enabling developers to pack up to 8 double precision or 16 single precision operands. The goal of introducing SIMD extensions to general purpose microprocessors was to accelerate DSP and multimedia applications at a small cost, traditionally ILP was extracted by increasing the dispatch logic and the number of functional units available, using SIMD instructions sets only the width of the functional units is increased. The problem with SIMD extensions is that the programmer and the compiler are responsible of extracting the instruction level parallelism (ILP) that previously was extracted by the out of order core of the microprocessor. As many multimedia and digital signal processing (DSP) algorithms work on data vectors, SIMD programming is well adapted to these classes of applications.

Amdahl's law

In regular situations a sequential algorithm need to be redesigned from scratch due to the effects of Amdahl's law which says "... the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude". This law is useful to measure the maximum expected improvement to an overall system when only part of the system is improved. When applied to parallel computing, this law allows us to estimate the theoretical maximum speedup to be attained when using multiple workers. Improvement is limited by the time needed to execute the strictly serial fraction of the algorithm which cannot be parallelized ($t_{seq} \in [0, 1]$). The time required by the sequential and scalar version of the algorithm t_1 on a single processor is $t_1 = t_{par} + t_{seq}$ where t_{par} represents the parallelizable fraction of the algorithm. Given $p \in \mathbb{N}$ processors to do a parallelizable work we have that:

$$t_p = t_{seq} + \frac{t_{par}}{p} = t_{seq} + \frac{t_1 - t_{seq}}{p}. \tag{5.1}$$

Normalizing (5.1) by t_1 we get $\tau_{seq} = \frac{t_{seq}}{t_1}$ where the acceleration (speedup) S_p for a parallelism degree of p is:

$$S_p \leq \frac{t_1}{t_p} \leq \frac{1}{\tau_{seq} + \frac{1-\tau_{seq}}{p}} \leq \frac{1}{\tau_{seq}}, \tag{5.2}$$

the reason to use **the less than or equal to** symbol is that the maximum speedup is only reached when the parallilizable work can be perfectly parallelized.

Considering now what happens when $p \to \infty$ where the maximum speedup is bounded by τ_{seq} as $S_\infty = \frac{1}{\tau_{seq}}$. Hence, the best parallel algorithm might not be the one which minimizes the total amount of computational work, but the one which reduces to the minimum the time it takes to perform the longest chain of tasks that must be performed *sequentially* or the *span* of the algorithm. **Figure 5-4** shows one of the first implications of Amdahl's law, as the degree of parallelism p grows the parallelizable work is distributed and the span of the algorithm converges to the size of the serial work that cannot be distributed. **Figure 5-5** plots the achievable speedups that can be obtained depending on the parallelizable portion of the algorithm.

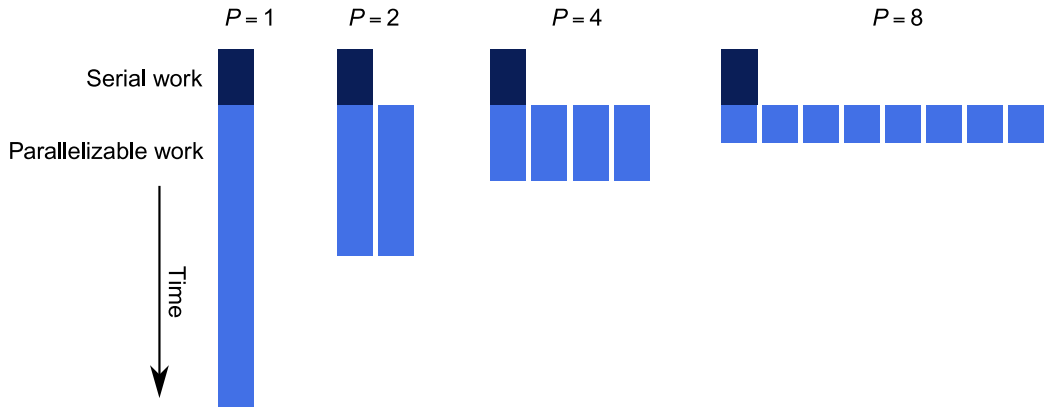


Figure 5-4: The expected speedup is limited by the non-parallelizable serial portion of the work (deep blue). Image taken from [93].

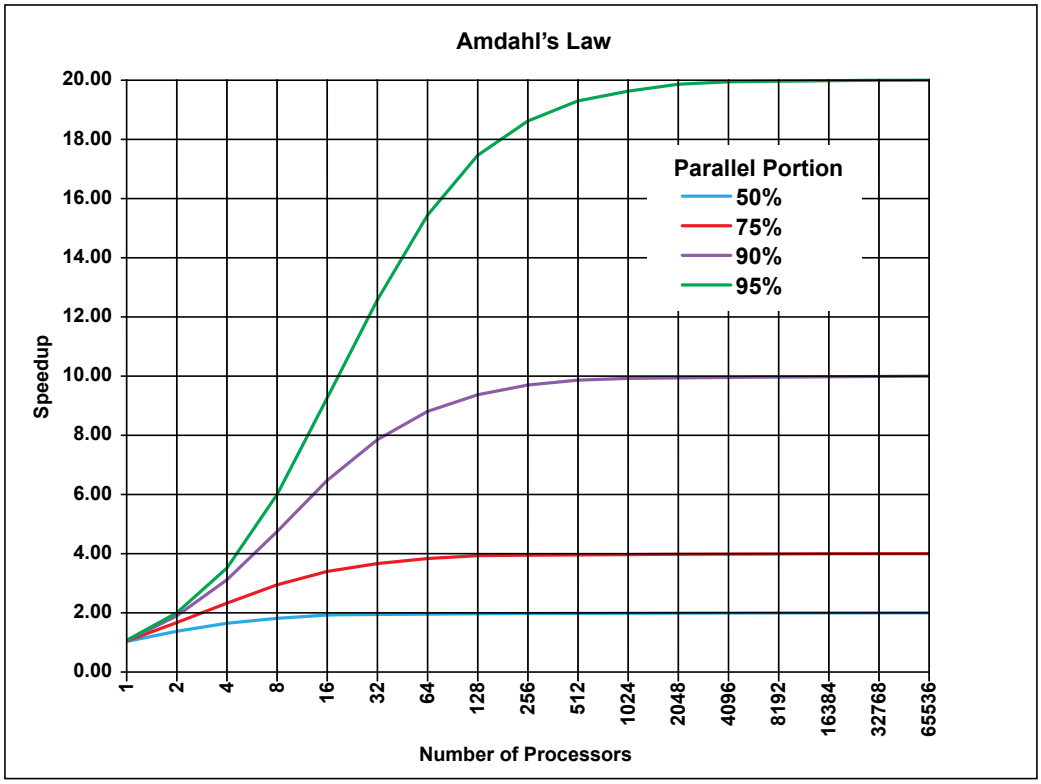


Figure 5-5: Amdahl's law (equation (5.2)) expresses how to calculate the theoretical maximum speedup of a program using multiple processors in parallel computing which is limited by the sequential fraction of the program. For instance, if 95% of the program can be parallelized, the theoretical maximum speedup would be 20x no matter how many processors are used. Image taken from Wikipedia

GPUs and other offload devices

An important recent trend is the development of attached processing units such as graphic accelerators and co-processors specialized in highly parallel workloads. Graphic Processing Units (**GPUs**) are promoted by their manufactures as Single Instruction, Multiple Threads (**SIMT**) architectures (a term coined by Nvidia), a category which corresponds to a **tiled SIMD** architecture formed by multiple SIMD processors. On a vector machine such as Cray1 or the modern MMX, SSE and AVX incarnations, there is a single processor or individual cores sequencing the instructions and some instructions are purely scalar executing on 8 to 64 bit data, but other instructions are vector and operate on multiple values (e.g. 512 bits of data can be interpreted as 16 single precision floating point numbers). In SIMT processors, each lane of vector computations can be interpreted as a separate thread, and a single hardware sequencer operates on a group (or block) of such threads allowing the parallel execution of thousands of them. A single instruction fetch is broadcast to all the individual processing elements, if a thread branches to a different directions, they are marked as diverged. The sequencer uses a mask to identify the threads that have diverged and then it fetches instructions and distributes it to all the processing elements enabled by this mask. Coherent code needs to be run on those blocks to maintain the efficiency of the entire block high. GPUs where originally designed for graphic applications particularly those with large amounts of fine-grained parallelism and exhibiting high coherence (a divergent control flow can significantly reduce the efficiency within a block) and as they have relatively small on-chip memories, latency is hided using an extremely large number of active threads.

Running computations on a co-processor or an accelerator is referred as **offload processing**, GPUs are not the only type of offload device available, other types of devices based on different technologies such as field programmable gate arrays (**FPGAs**) or co-processors formed by many-core processors exist. A nice example of the last type is given by the Intel MIC (Many Integrated Cores) architecture, which contains more cores than traditional CPUs (over 50) and allows the developers to create platforms running at trillions of calculations per second using the fast and familiar Intel Xeon processor and the Intel Xeon Phi coprocessor, each core has wide vector units able that are able to execute 16 simultaneous single-precision floating point operations, the only condition is to have good data locality on the implementation algorithm because the cache size by core is very small (L1 32K instruction and data caches and L2 512K cache per core). Other semiconductor companies are now commercializing many-core processors systems *e.g.* Tilera with its Tile GX processors family[3] and Kalray with their MPPA products which can be used either as a stand-alone embedded solution or as an acceleration co-processor of a host CPU.

Offload devices (such as GPUs) are usually located on the PCIe bus they do not share memory with the host and data must be transferred across the PCIe bus to a memory local to the offload device before it can be processed. To alleviate this problem, some manufacturers have begun to integrate those units into the same die as the main processor cores, this is the case of AMD and Intel. NVIDIA (the most important GPU manufacturer) also makes integrated CPU/GPUs using ARM cores for the embedded and mobile markets. The advantage is that memory is physically shared by the GPU and CPU processors.

5.1.3 OpenMP

A single-threaded program can be converted easily into a multi-threading one without worrying about the API's environment variables and other small details using OpenMP. OpenMP is an API specification created in 1998 by a committee of vendors to allow and easy specification of shared-memory concurrency in Fortran, C and C++ programs. Todays most important compilers support OpenMP that includes GNU GCC compiler, Intel C/C++, Microsoft Visual C/C++, Clang and LLVM. So it works on almost every modern operating system such as Windows, Linux and OSx. Different versions the API specification exists so programmers are suggested to verify first which versions are supported by each compiler.

OpenMP consists of API's pragmas and several other settings to control OpenMP-specific environment variables. Pragma directives offer a way for each compiler to offer machine and operating system-specific features while retaining compatibility with the Fortran, C and C++ languages. According to the ANSI C and C++ standards, if a compiler does not recognize a given pragma, it must ignore it, this makes completely safe to place OpenMP pragmas in code without worrying if the code will compile on a different toolset. When a master thread arrives to a declared parallel region (by an OpeMP pragma), it forks a specified number of slave threads to divide a task among them (using a fork-join model). Parallel regions are declared to control how code is assigned to threads, the default behavior makes all threads within the team wait at the end of the parallel region until all other threads have finished running before being joined back again so that the main thread can resume the execution of the code in sequential mode. This behavior can be modified but needs to be explicitly stated by the programmer.

Initially, OpenMP does not guarantee how many threads will be created, usually, it choses a number equivalent to

[3] `http://www.tilera.com/products/processors/TILE-Gx_Family`

available number of execution pipelines. On standard multiprocessor system this number would be the number of cores available, but systems equipped with Hyper-Threading Technology, the number of pipelines is twice the number of processors.

All OpenMP pragmas are of the form `#pragma omp`. This directive is adapted by more optional clauses that modify the meaning of the construct, for example, a parallel region is defined using the `parallel` construct to form a single construct (*i.e.* `#pragma omp parallel`).

When the master thread encounters this pragma, a team of threads is forked and the single statement or the block statements (enclosed within curly braces) within the parallel region are executed on each thread, threads are joined after the last statement in the region. The following code

```
1  double k1 = alpha;
2  double k2 = (1 - alpha);
3  #pragma omp parallel for
4  for(int i = 0; i<n; i++)
5  {
6    C[i] = k1 * A[i] + k2 * B[i];
7  }
```

calculates the linear combination of two vectors `A` and `B` component by component and stores the results in `C`. Instead of executing each iteration sequentially, the iterations of the loop (the values from $i = 0$ to $i = n - 1$) are divided amongst the threads of the team initiated in line 3.

OpenMP offers many other pragmas used to identify code blocks to threads, to control the scope of the variables to be shared across threads or to keep local inside the individual threads, to specify barriers and force the synchronization of threads, how to schedule the tasks (for task-level parallelism) or loop iterations to threads and so forth.

Memory layouts and data reorganizations to enforce spatial locality

Code transformations are not the only possibility to improve performance, a different alternative is to modify the **memory layout** of the referenced data. Changes in data layout affect how data structures are declared and addressed, this consideration is even more important for parallel computing because a reorganization of the data may help obtain the optimal performance in the vectorization.

An array containing a contiguous collections of data items is one of the most common and well-known data structures. The usual approach to data abstraction is to declare structures to represent the properties and states of an object. Instances of an object are created in runtime and organized in collections of that structure. These type of collections are popularly known as **Array of Structures** (*AoS*) layout. The problem with this layout is that it does not align well for vectorization or caching.

An alternative layout is to create a collection of each element or state in the structure, this layout is known as **Structure of Arrays** (*SoA*). Vectorization in this layout is normally easier since the same function is replicated over every element (or groups of elements) in the elemental array. This type of processing pattern is popularly known as **map**, the function being replicated is called the **elemental function** since it is applied to all the elements of a collection, usually producing a

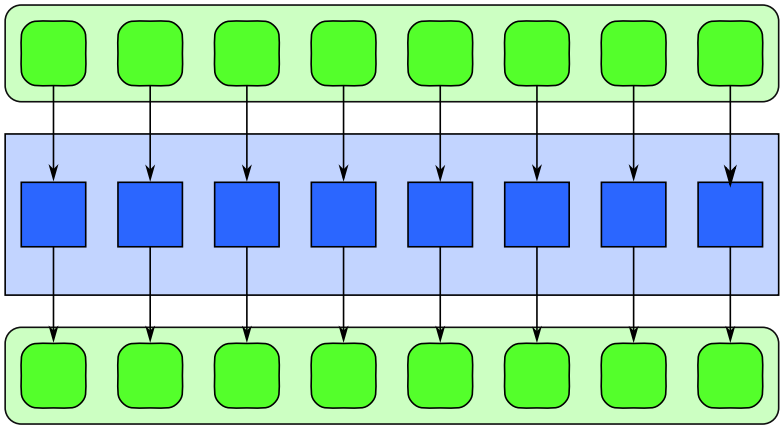


Figure 5-6: Map pattern, a function is applied to all the elements of a collection, usually producing an output with the same shape as the input. Image taken from [93].

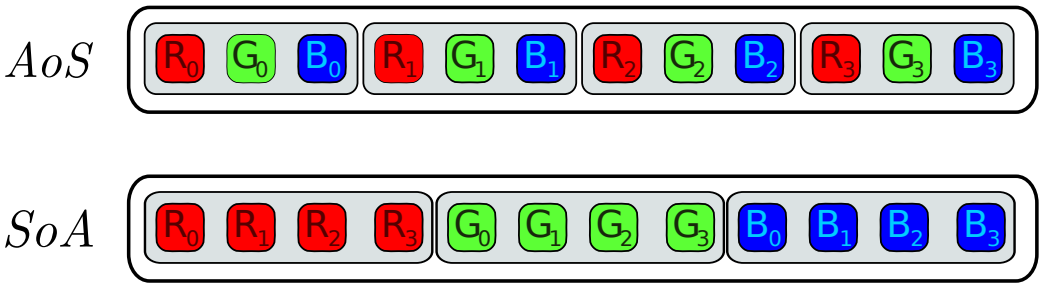


Figure 5-7: Array of structures (*AoS*) versus structure of arrays (*SoA*). The *SoA* form is typically better for vectorization. However, in some circumstances such as when data is accessed incoherently, *AoS* may lead to better cache utilization.

new collection with the same shape as the input (see **Figure** 5-6).

To illustrate the differences between both data structure organizations an example is proposed: suppose that we can compare the *AoS* and *SoA* arrangements for storing the r,g and b components of a set of points in an RGB image, AoS requires the declaration of an array formed by multiple RGB elements (see **Listing 1**). In contrast, the *SoA* implementation is composed by a set of arrays each one containing an individual component and example is provided in **Listing 2**. A loop visiting all the components of an RGB point before moving to the next point using the *AoS* arrangement exhibits a good data locality of reference because all elements in the lines fetched to the cache memory are utilized. In contrast, a loop that just visits one component of all points (say the red component in our example) has a less satisfying locality of reference because many of the elements in the fetched cache lines are not used. Furthermore, a common disadvantage of the *AoS* arrangement is that each individual reference in a loop exhibits a non-unit stride access pattern, additionally, this type of arrangements may not be very vectorization friendly due to the non-sequential access patterns that may produce long instruction latencies.

The selected type of arrangement may have an important impact on the performance of the application. An hybrid arrangement can be created as an alternative to *AoS* and *SoA*, this approach is known as Hybrid Structure of Arrays (HSoA). For the RGB images example, the HSoA arrangement is shown in **Listing 3**.

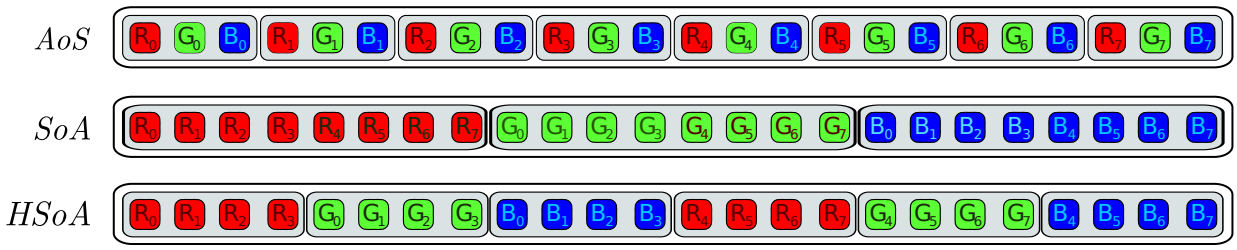


Figure 5-8: Three alternative arrangements for storing RGB images can be considered: the Array of Structures (*AoS*) (top), the Structure of Arrays (*SoA*) arrangement (middle) and the Hybrid Structure of Arrays (*HSoA*) (bottom).

```
// RGB points in AoS arrangement
struct {
  float red;
  float green;
  float blue;
}AoS_rgb[200];
```

Listing 1: *AoS* data layout.

```
// RGB points in SoA arrangement
struct {
  float red[200];
  float green[200];
  float blue[200];
}SoA_rgb;
```

Listing 2: *SoA* data layout.

```
// RGB points in HSoA arrangement
struct Points{
  float red[4];
  float green[4];
  float blue[4];
}Hybrid_rgb[50];
```

Listing 3: HSoA data layout suitable for SIMD manipulation, here the vector cardinality is four.

In the HSoA arrangement, the *SoA* arrays are divided forming an array of structures containing n/m-elements, each instance of this structures contains a set of sub-arrays of width m for each of the items in the *AoS* arrangement, the value of m is often referred as the *cardinality* of the SIMD register.

The differences between all three approaches is easily visualized in **Figure 5-8**. For the case of out RGB example, HSoA is well adapted for the vectorization of a number of algorithms such as color gradient and feature computation, color-space conversions, etc. Elements should be aligned and arranged according to their data type sizes and the correspondent vector register widths: architectures equipped with SSE4a technology use 128-bit registers which allow to allocate sixteen 8-bit fixed point values, four 32-bit single precision floating point values and just two 64-bit double precision floating point values.

Hand optimization of the data structures is often required, the baseline or *out of the box* version of an application is often developed first and only after the application executes the algorithm correctly and produces the expected results profiling is run to detect bottlenecks. Respecting this procedure is very important to save us from the arduous and fruitless job of optimizing irrelevant structures and sections of code by hand. Some of the most popular and robust profiling tools available are VTune, Gprof and Oprofile. It is worth emphasizing that the *SoA* layout is not ideal in all circumstances, specially when accesses are random or incoherent, unneeded data might be brought into the cache.

5.2 Covariance tracking implementation

Much of the research work of this thesis formed part of the European project ITEA/Spy which demanded a robust real-time object tracking algorithm targetted to run on embedded low-power consumption systems. This section explores all the different techniques that were evaluated in this work to accelerate the computation of the covariance descriptor in order to achieve this objective.

5.2.1 Image processing algorithmic and architectural optimizations

Computer vision and image processing algorithms are often divided into three different abstraction levels:

- **Low-level**: They execute regularly and independently of the data stored inside the data arrays. For the case of image processing algorithms, the same operator is applied for all the picture elements or image locations irrespectively of their values. Algorithms of this type receive an input image and create and output a new one (or a set of them). Examples: filtering algorithms (*e.g.* linear, no-linear, recursive, non-recursive) such as smoothing and gradient filters.

- **Intermediate-level**: Algorithms of this type are applied to analyze the information contained on the input data arrays (*e.g.* pixel values and their neighborhoods for the case of image processing), to extract semantic information: feature point, contours, background subtraction and object detectors. This type of image processing algorithms usually receive one or more input images and produce high-level (or structure) information.

- **High-level**: These algorithms are charged of providing an interpretation and taking decisions based on the information provided by low-level and intermediate-level algorithms. Typical computer vision algorithms of this type are: object tracking, activity recognition, scene understanding, etc.

Most of the research on algorithm-architecture adaptation for image processing applications is concentrated on the low and intermediate levels. **Figure 5-9** shows the data flow followed by the covariance descriptor computations and matching algorithm. Each pixel on the initial image is used to calculate a set of n_F feature images (primary features), this step involves the computations of gradients, color transformations, bilinear interpolations and/or LBP texture operators. The next steps consist in calculating the n_P feature crossed products (or second order features) and the integral images from all the primary and second order features, that makes $n_F + n_P$ integral images in total. Lets calculate now the amount of memory space required by the covariance matching algorithm, as $n_P = n_F(n_F + 1)/2$, for the case of the ELBCM$_{Lum}$ descriptor we have $n_F = 7$ primary features and $n_P = 28$ secondary features that makes 35 integral images in total. For

a one megapixel image of size 1024×1024 and using double precision floating point arithmetic (which needs 8 bytes per value) to compute the integral images the algorithm would require $1024 \times 1024 \times 35 \times 8 \approx 293$ MB of memory!. All this transformations apply low level algorithms where the input image is transformed to create a set a new image (or a set of images), a set of algorithm optimizations are proposed in this section to improve the algorithms runtime performance, the set of notations common to all the different versions of the algorithm is given here first. The input image I whose height and width is represented by h and w, is first decomposed by the $\phi(I, x, y)$ operator into a collection of n_F feature images stored in F. From them n_P images of the crossed-feature products (*i.e.* $n_P = n_F(n_F + 1)/2$) are obtained and stored in P. **Table 5.2** summarizes all these notations.

Table 5.2: Covariance descriptor algorithm notations.

Notation	Meaning
I	Input image
h and w	Height and width of I
n_F	Number of features used to build the descriptor
n_P	Number of crossed-products of features $n_P = n_F(n_F + 1)/2$
F	A data structure that contains all the features images
P	A data structure containing all the feature image products
I_F and I_P	The summed area tables (integral images) computed from F or P

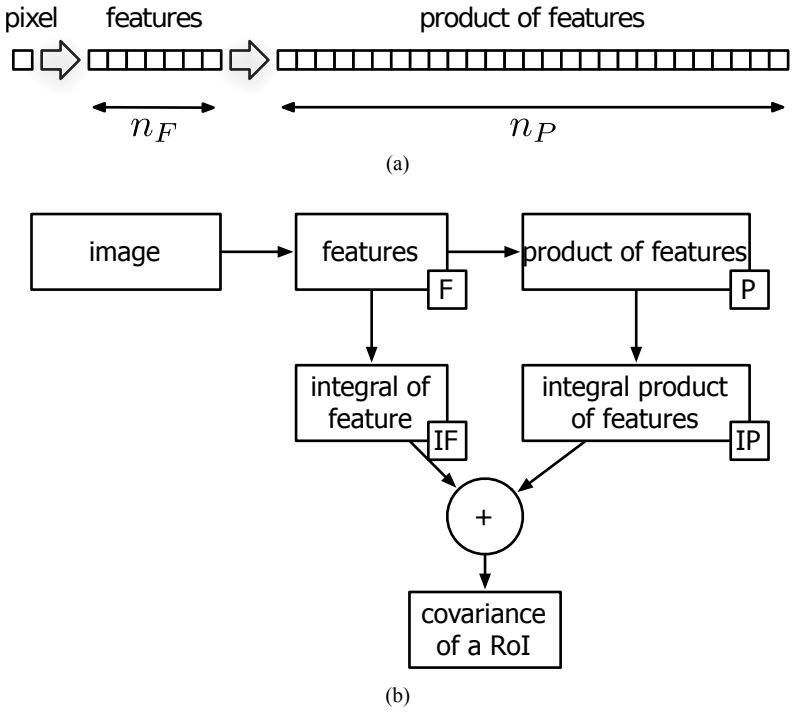


Figure 5-9: Covariance data flow: features computation

5.2.2 Integral images accuracy considerations

Before describing the proposed optimizations for the parallelization and vectorization of the covariance descriptor algorithm, it is necessary to study its correctness limits related to the data-type precision used during the implementation of the algorithm. The fewer the bits used to represent the data, the more operations it is possible to perform in parallel using SIMD instructions, but there is a trade-off decision here because a reduction on the precision bits may cause a reduction of performance in the best case or meaningless descriptors in the worst scenario. Floating point data types facilitate the programming effort, but it is extremely important to be aware of its pitfalls.

In the covariance descriptor computation data-flow in **Figure 5-9**, features are obtained from the original luminance (gray-scaled) or colored images (RGB). Each feature represents a different aspect of the image and each one has its own dynamic range. Integral images are computed from them and from their crossed products. To prevent us from computing spurious or meaningless descriptors it is important to assure that the data type used during the accumulation all those values is able to calculate the accumulation accurately. This section analyzes different implementations for the integral image calculation and their implications.

Integral images are a type of two dimensional data reduction or more precisely a scan collective operation which produces a set of partial reductions of the input image forming a new output image of the same size.

The finite number of bits used to real numbers requires to accept just an approximation of the true value. The precision of a number is measured in terms of the number of digits that contain meaningful data (referred as the significant digits). Depending on the type of data being used our algorithms are able to accumulate a different number of elements without incurring in accuracy errors:

- **Integer pixel values**: The simplest case corresponds to the accumulation of b-bit integers in m-bit valued integral images. In this scenario it is possible to accumulate at most 2^{m-b} values without accuracy errors.

  Example: If the image is composed by 8-bit integers and they are accumulated on integral images represented by 32-bit floating point (IEEE754) values which have 24-bits[4] to store the significand, then we can store $2^{24-8} = 65536$ pixels of type byte without accuracy errors. The double precision 64-bit floating point format with 53-bit to store the significand allows us to accumulate much more pixel values ($2^{53-8} = 35.1843 \times 10^{12}$).

- **Fixed point values**: In the covariance descriptor algorithm we require to take the product of the features, data with pixed point values are of the order of the square of feature image pixels (*e.g.* the square of a 16-bit fixed point values has a range of 32 bits). A double precision integral image can hold up to $2^{53-32} = 2097152 \approx 2$ million of such 32-bit values without loss of precision. This corresponds to a 1400×1400 square image.

- **Floating point values**: Because of their flexibility, floating points values are preferred to accelerate development time. Here we are interested in verifying the correctness and the applicability limits of floating point feature images in the covariance descriptor algorithm. What matters in the integral image is the ***fixed point precision*** of its data type, this is equivalent to the number of possible values data can take within a certain range. The addition of floating-point numbers is different from binary integer addition (or fixed-point addition). Fixed point addition is more or the less analogous to how additions are performed by hand: numbers are lined up one over the other and additions are

[4] Only 23 bit are explicitly stored but a 24-th bit is implied by the format.

performed one digit at a time starting from the least significant bits. Floating point addition runs a similar procedure too, but first floating point numbers are shifted so that the corresponding digits line up. If the two numbers have significantly different exponents then some of the least significant digits are discarded.

Floating point arithmetic has many pitfalls: a real value such as $(0.1)_{10}$ in base ten has no exact finite representation and its (approximated) binary form is: $(0.1)_{10} = (0.000110011001100\cdots) \approx 0 : 10000000149011611938476562 5$. Moreover, floating point arithmetic does not respect neither the distributive nor associate operators

$$(-10000001 + 10^7) + 0.5 \neq -10000001 + (10^7 + 0.5), \tag{5.3}$$

this attributed to absorption and cancellation phenomena exemplified by **Listings 4** and **5**.

```
1  float a = 1.0e7;
2  float b = 0.5;
3  float c;
4
5
6
7  // = 10000000.0 and not 10000000.5
8  c = a + b;
```

Listing 4: *Absorption* phenomena.

```
1  float a = 1.0;
2  float b = 1.0e-7;
3  float c = 1.0;
4  float d = 1.0e7;
5  float e;
6
7  // e = -1.192 and not -1
8  e = ((a - b) - c) * d;
```

Listing 5: *Cancellation* phenomena.

Large summations are prone to run out of bits very fast, if a naive serial algorithm is used the partial sum can grow very large in comparison to the new values to be added to it and if the new values to add are less than the smallest representable increment the summation will not have any effect on the accumulator and the increments are rounded to zero. This is equivalent to ignoring some pixels during the integration process which obviously conduces the algorithm to wrong results. Line `8` in **Listing 4** shows how all the information from one summand is lost because the summand `a` is much more larger than `b`. This phenomenon might be difficult to see when summing large sets of floating-point numbers because usually such pronounced differences are no seen between the individual terms, but errors can still grow from the accumulation of many small errors. **Listing 4** is an eloquent example of how precision is quickly lost when the magnitude of the two numbers to be added is significantly different. In summation area tables (such as the integral images are), one of the operands represents the accumulated term so far and in general it can be much larger than the terms being added to it. If the data spans over an extreme range of floating point values say from 2^{-100} to 2^{100} for example, storing the data with fixed point precision would require 200 bits per pixel. Neither a single nor a double precision valued integral image is sufficient to hold these values without error. This problem is usually disregarded because usual data rarely span such a large ranges of precisions. Integral images are applicable in most cases, but single precision integral images need to be handled with care and avoided unless we are very sure of not incurring in precision errors.

For an image X its integral image I at the pixel (x, y) is the sum of all the pixels values of X above and to the left of it. This is expressed mathematically as:

$$I(x, y) = \sum_{p=0}^{x} \sum_{q=0}^{y} X(p, q). \tag{5.4}$$

Equation (5.4) is implemented by **Algorithm 12** using two loops for iterating within the image pixel elements. A different type of nation is used here, the pair of images $X(x, y)$ and $I(x, y)$ is now denoted by $X[\mathtt{i}][\mathtt{j}]$ and $I[\mathtt{i}][\mathtt{j}]$ where the spatial location (x, y) is now denoted by $[\mathtt{i}][\mathtt{j}]$, where `i` represents the index of the lines and `j` indexes the columns. These is the type of notations used consistently throughout the rest of the chapter to explain other image processing algorithms making them compatible with popular notations used for linear algebra and image processing algorithms in C (refer to Numerical Recipes in C [111]). The relation $I[\mathtt{i}][\mathtt{j}] = X[\mathtt{i}][\mathtt{j}] + I[\mathtt{i}][\mathtt{j}-1] + I[\mathtt{i}-1][\mathtt{j}] - I[\mathtt{i}-1][\mathtt{j}-1]$ at line number 3 reuses the previous calculations on I. Pixel $X[\mathtt{i}][\mathtt{j}]$ is added to the already integral image computed values at the left $I[\mathtt{i}][\mathtt{j}-1]$ and above $I[\mathtt{i}-1][\mathtt{j}]$, the value at diagonal, $I[\mathtt{i}-1][\mathtt{j}-1]$ needs to be subtracted once because it is accumulated in both directions. All the values at $I[\mathtt{i}][-1]$ and $I[-1][\mathtt{j}]$ are initialized with zeros. For every computed pixel, this algorithm makes 3 `ADD`s and (4 `LOAD`s + 1 `STORE`) = 5, so its arithmetic intensity[5] is $AI = 3/5 = 0.6$.

Algorithm 12: Speed-optimized integral image computation.

Data: An image X of size $h \times w$.
Result: The integral image I of size $h \times w$.

```
1 for i ← 0 to h − 1 do
2     for j ← 0 to w − 1 do
3         I[i][j] = X[i][j] + I[i][j − 1] + I[i − 1][j] − I[i − 1][j − 1]
```

One pitfall of **Algorithm 12** is that as it progresses in its walk across the image, the magnitude of the values accumulated on I may rapidly grow in comparison to the values on X, if the partial sums stored in I grow large enough, new values to add from X will be less than the smallest representable increment and their summation will have no effect. A two-pass algorithm is thus proposed to alleviate this situation, all the accumulations are now separated into two different passes: one vertical and one horizontal. If the accumulations are executed first in the vertical direction, $S_y[\mathtt{i}][\mathtt{j}]$ accumulates all the pixels in the `j`-th column of X from line 0 to line i. Conversely, the accumulator $S_x[\mathtt{i}][\mathtt{j}]$ can be used in the place of S_y if the accumulations are executed in the horizontal direction first. Following this approach, the magnitudes differences between the operands to add and the running accumulator are reduced together with the chances of incurring in accuracy errors (see **Figure 5-10**). The S_y accumulator of the two-pass integral image algorithm (passing in the vertical direction first) is represented mathematically as:

$$S_y(x) \quad = \quad \sum_{p=0}^{y} X(x, p). \tag{5.5}$$

Where $S_y(x)$ calculates the vertical accumulations of all the terms in column x from line zero up to line y. The final accumulations are performed in the second pass as:

$$I(x, y) \quad = \quad \sum_{q=0}^{x} S_y(q). \tag{5.6}$$

[5] The ratio between the arithmetic complexity and the number of memory accesses.

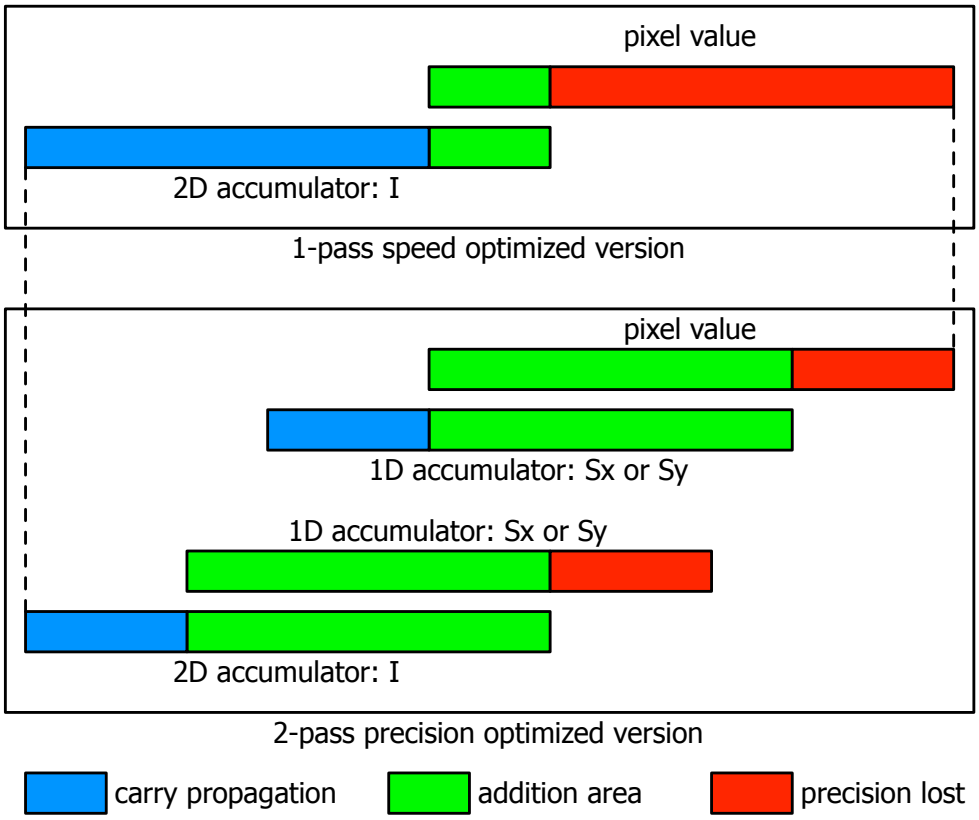


Figure 5-10: Due to the limited number of bits, the 1-pass speed optimized version (**Algorithm 12**) may sacrifice the accuracy of the calculations in favor of the speed. The magnitude of the values accumulated on I may rapidly grow in comparison to the values on X, if the partial sums stored in I grow large enough, new values to add from X will be less than the smallest representable increment and their summation would have no effect. The 2-pass version in (**Algorithms 14 or 13**) initially separate the accumulations in the vertical or horizontal directions (S_y or S_x). In the second pass, their values are accumulated, as their magnitudes are more likely to be similar, the 2-pass version reduces the number of precision bits lost.

Two different version of the two pass algorithm exists depending on the initial direction of accumulation (*i.e.* vertical or horizontal), **Algorithms 13** and **14** show both versions. As the number of for loops is doubled, the two-pass integral image is slower than the single pass (speed-optimized method) and each pass inside the loop demands one addition (`ADD`), two load operations (`LOAD`) and one store (`STORE`). In consequence, the arithmetic intensity goes down to $AI = (2\texttt{ADD})/(4\texttt{LOAD} + 2\texttt{STORE}) = 0.333$. An this analysis without considering the lack of spatial locality of the data when computing the vertical accumulations, in summary, the two-pass accumulation method sacrifices the execution speed in favor of the accuracy of the accumulations.

Algorithm 13: Vertical accumulation 2-pass method.

Data: An image X of size $h \times w$.

Result: The integral image I of size $h \times w$.

1 $S_y[-1 \cdots w-1][-1 \cdots h-1] \leftarrow 0$
2 **for** $i \leftarrow 0$ **to** $h-1$ **do**
3 **for** $j \leftarrow 0$ **to** $w-1$ **do**
4 $S_y[\mathtt{i}][\mathtt{j}] \leftarrow S_y[\mathtt{i}-1][\mathtt{j}] + X[\mathtt{i}][\mathtt{j}]$
5 **for** $i \leftarrow 0$ **to** $h-1$ **do**
6 **for** $j \leftarrow 0$ **to** $w-1$ **do**
7 $I[\mathtt{i}][\mathtt{j}] \leftarrow S_y[\mathtt{i}][\mathtt{j}-1] + S_y[\mathtt{i}][\mathtt{j}]$

Algorithm 14: Horizontal accumulation 2-pass method.

Data: An image X of size $h \times w$.

Result: The integral image I of size $h \times w$.

1 $S_x[-1 \cdots w-1][-1 \cdots h-1] \leftarrow 0$
2 **for** $i \leftarrow 0$ **to** $h-1$ **do**
3 **for** $j \leftarrow 0$ **to** $w-1$ **do**
4 $S_x[\mathtt{i}][\mathtt{j}] \leftarrow S_x[\mathtt{i}][\mathtt{j}-1] + X[\mathtt{i}][\mathtt{j}]$
5 **for** $i \leftarrow 0$ **to** $h-1$ **do**
6 **for** $j \leftarrow 0$ **to** $w-1$ **do**
7 $I[\mathtt{i}][\mathtt{j}] \leftarrow S_x[\mathtt{i}-1][\mathtt{j}] + S_x[\mathtt{i}][\mathtt{j}]$

A variation of the two-pass accumulation method for the integral images calculation can be obtained using the following recurrence relation

$$\begin{aligned} S_y[\mathtt{i}][\mathtt{j}] &= S_y[\mathtt{i}-1][\mathtt{j}] + X[\mathtt{i}][\mathtt{j}] \\ I[\mathtt{i}][\mathtt{j}] &= I[\mathtt{i}][\mathtt{j-1}] + S_y[\mathtt{i}][\mathtt{j}], \end{aligned} \tag{5.7}$$

where each pixel $X[\mathtt{i}][\mathtt{j}]$ is added (vertically) to its corresponding element in $S_y[\mathtt{i}-1][\mathtt{j}]$ and the image $S_y[\mathtt{i}][\mathtt{j}]$ contains all the vertical accumulations. The integral image at $I[\mathtt{i}][\mathtt{j}]$ is computed using the vertically accumulated values in $S_y[\mathtt{i}][\mathtt{j}]$ and the integral image pixel at the left $I[\mathtt{i}][\mathtt{j}-1]$. **Algorithm 15** implements this idea, it is not necessary to create a complete image to handle the values of the vertical accumulations $S_y[\mathtt{i}][\mathtt{j}]$, just a pair of arrays $S[\mathtt{j}]$ and $S_{prev}[\mathtt{j}]$ of size w is enough. This last pair of lines is swapped (just their initial pointers) at the end of each iteration (see **Algorithm 15** line 8). This variation of the 2-pass accumulation algorithm is equivalent (*i.e* the elements in $X[\mathtt{i}][\mathtt{j}]$ and the intermediate values are accumulated in exactly the same way) the difference is that this method runs in a single pass, requires less memory (only two intermediate arrays that are swapped) and only requires two additions (`ADD`), three loads (`LOAD`) and two stores (`STORE`). So, its arithmetic intensity is $AI = 2/5 = 0.4$. In conclusion, this method offers a good compromise between the advantages of the speed-optimized method and the precision-optimized two-pass method.

Algorithm 15: 1-pass integral image computation algorithm optimized for speed and accuracy.

Data: An image X of size $h \times w$.
Result: The integral image I of size $h \times w$.

1 $S_y[-1 \cdots w-1] \leftarrow 0$
2 $S_{y_{prev}}[-1 \cdots w-1] \leftarrow 0$
3 **for** $i \leftarrow 0$ **to** $h-1$ **do**
4 **for** $j \leftarrow 0$ **to** $w-1$ **do**
5 $S_{y_{tmp}} \leftarrow S_{y_{prev}}[\mathtt{j}] + X[\mathtt{i}][\mathtt{j}]$
6 $S_y[\mathtt{j}] \leftarrow S_{y_{tmp}}$
7 $I[\mathtt{i}][\mathtt{j}] \leftarrow I[\mathtt{i}][\mathtt{j}-1] + S_{y_{tmp}}$
8 **swap**(S_y,$S_{y_{prev}}$)

For the case of the feature images used to compute the covariance descriptor, we can assume pixels to be in the range 0 to 255 which requires 8-bit to represent them. The single precision floating point data type (`float` 32-bit) has 24 bits for

the significand, thus a total of:

$$2^{24-8} = 65,536$$

values can be summed without accuracy problems using the single pass version of the algorithm. Equivalently for the double precision floating point values (`double` 64-bit), there are 53 bits for the significand which can store a total of:

$$2^{53-8} = 35.1843 \times 10^{12}$$

values.

These values are enough for calculating integral images of maximum sizes 256×256 for the single pass method and $65,536 \times 65,536$ for 2-pass method using single floating point precision. For the case of double precision data types, its possible to accumulate a totality of $(5.931641 \times 10^6) \times (5.931641 \times 10^6) = 35.1843 \times 10^{12}$ using the single pass method.

The real limitation appears when dealing with the integral images of second-order degree features (feature crossed-products). For these images the range of values goes from 0 to $2^{16} = 65,535$ (supposing 8-bit fixed point feature images), the maximum number of pixels we can accumulate using single precision floating point integral images for these crossed-products is:

$$2^{24-16} = 256,$$

which limits us to patches of size 16×16 for the single pass accumulation method, and 256×256 for the 2-pass accumulation method. For the case of double floating point precision the algorithm can handle:

$$2^{53-16} = 137,438,953,472$$

values of this type, which means that images of size $370,727 \times 370,727$ can be represented adequately enough by double precision data images using the single pass accumulation method. For the type of feature images being used, single precision floating point integral images are clearly not enough to guarantee the correctness of the algorithm when handling images bigger than 256×256 due to insufficient number of bits used to handle precision in the significant.

Concerning the implementation of the covariance descriptor algorithm, the necessity of recurring to double precision is an undesired constraint. Not only do 64 bit operations usually have a bigger latency, but more importantly, the degree of SIMD parallelism is reduced by two (without considering the overhead of handling 64-bit data instead of 32-bit). Furthermore, the set of Neon SIMD instructions does not include 64-bit instructions.

Some alternatives exist that might be useful to overcome or reduce the impact of this problem:

- Considering the possibility of not to use integral images. This option surely has an strong impact on the speed of the algorithm when comparing multiple covariance descriptors (as it is the case for tracking applications).
- If all the inputs are about the same size, the parallel **inclusive scan** pattern implementation represented in **Figure** 5-11 (right) offers a possible solution since the intermediate results have a bigger chance to have similar values. This arrangement (detailed in [93]) is advantageous as it allows us to achieve some degree of parallelism, but it is not perfect either, indeed it is quite easy to invent input values that break any specific ordering of the operations.
- Using compensated summations [63], for instance, the **Kahan summation algorithm** which significantly reduces

the numerical error by keeping a separate running compensation, which is a variable used to accumulate the small errors.

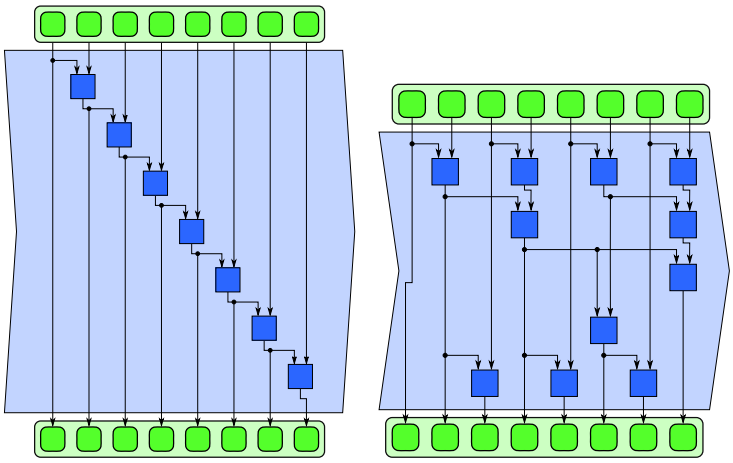


Figure 5-11: Serial and parallel implementations of the inclusive scan pattern. Image taken from [93].

5.2.3 Covariance tracking algorithm baseline analysis

In the baseline version of the algorithm the complete set of feature images F is stored separately using a cube data structure (represented in code by `fmat[k][i][j]`) which can be regarded as an instance of a *structure of arrays* (*SoA*) data structure. The index `k` is used here to select the desired feature image and the pair (`i,j`) to select the spatial coordinates. Image cubes are straightforward to implement, the required arithmetic to compute the memory a address using a table of 3D pointers only demands three integer additions, still, the latency time of a memory access is extremely dependent on the data access pattern.

Two different versions of the algorithm were benchmarked:

1. **Region of interest (ROI)**: the algorithm uses previous target position information to estimate a confined tracking zone where the algorithm is convinced that target is located.

2. **Full image**: all feature computations and integral images are calculated uniformly for every pixel location on the input image I.

The ROI version of the algorithm is suggested for single target tracking applications, as it reduces significantly the computing effort, but as the number of concurrent targets grows it is wiser to operate uniformly over the complete image. Both versions were benched using VTune Amplifier XE 2013 using a Nehalem architecture (Intel Core i5) with 4 GB of RAM and running Ubuntu Linux 13.04. The profiling results for these baseline implementations are shown in **Table 5.3** using the Panda and Pedxing3 sequences already presented in the previous chapters.

In the ROI implementation, function `lbp_r1u8_operator_f32` (responsible of applying the LBP operator to compute the pattern angles to introduce in the ELBCM feature vector) takes about 51.98% of the execution time, the second most consuming function (32.68% of the execution time) is `sumMatrices_SOA`, this function computes the kernel of the covariance descriptor algorithm. Based on these results the priority should be given to `lbp_r1u8_operator_f32`, however, in

Table 5.3: Baseline scalar *SoA* algorithm profiling results for ROI and full image results.

Function name	**Description**	ROI		Full-image	
		CPU Time	%	CPU Time	%
`lbp_r1u8_operator_f32`	LBP operator	3.235s	51.98%	2.910s	9.94%
`sumMatrices_SOA`	I_F and I_P computation	2.034s	32.69%	25.616s	87.51%
`rgb8matrix_get_L_RGB_F32`	Gets L,R,G and B features	0.492s	7.91%	0.504s	1.72%
`bilinearInterpol`	Bilinear interpolation	0.462s	7.42%	0.243s	0.83 %

the full-image results it appears that the execution speed of the algorithm is considerably affected by the number of pixels being treated, here, function `sumMatrices_SOA` consumes about 87.50% of the application running time becoming the most critical function to optimize.

The kernel function (`sumMatrices_SOA`) can be decomposed into three different stages:

1. point-to-point features crossed products,
2. the integral image computation of features,
3. the integral image computation of products.

The first stage is by far the most consuming one, here, the set of image features F (`fmat[k][i][j]` in code) is accessed one-by-one for a particular spatial location $(\mathtt{i}, \mathtt{j})$. As the set of features are located different images, there is no spatial locality, nearby locations in a particular feature image are loaded to fill the cache lines, but these values are immediately discarded and prefetching hardware is underutilized. This obviously traduces into large amounts of cache misses that cause several CPU stalls that increase the application running time.

Three different methods are projected to optimize the covariance tracking CT running time using desktop and embedded processors with parallel architectures. In rough terms they are based on:

1. a data layout transformation (*SoA→AoS*),
2. architectural optimizations such as:
   - multi-threading the *SoA* version with OpenMP middle-ware,
   - using SIMD instructions (SSE/AVX for Intel, Neon for ARM),
3. a loop-fusion and serialization transformations code transformation.

5.2.4 *SoA→AoS* transformation

The goal of *SoA→AoS* data layout transform (Structure of Arrays to Array of Structures) consists in transforming a set of independent arrays into one single array where each cell is a structure combining the elements of each independent array. The contribution of such a transform is to leverage the cache performance by enforcing spatial and temporal cache locality.

The first aspect we want to optimize is the locality of the features for a given point of coordinates (i, j). In the *SoA* version we have two cubes: one that stores all the pixel features F_{SoA} (`fmat` in code) which size is $n_F \times h \times w$ and a

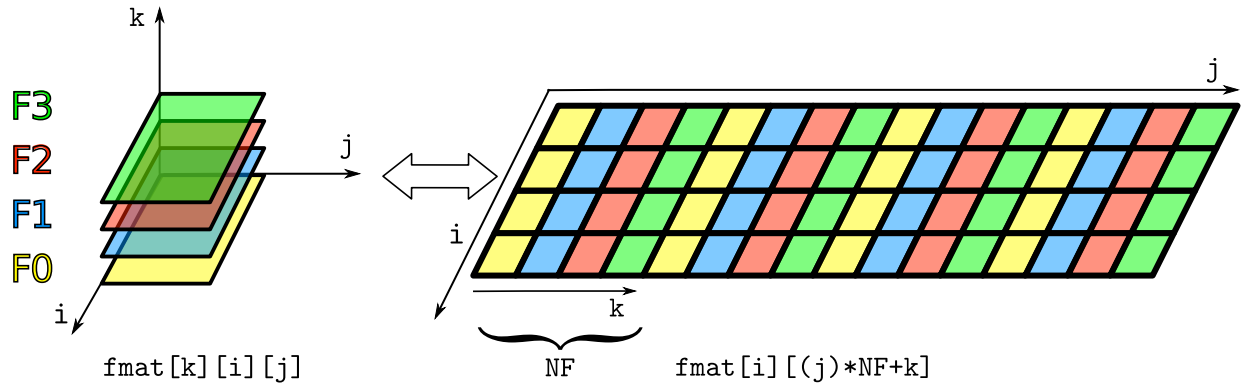


Figure 5-12: Layout of *SoA* cube and the *AoS* feature matrix.

different cube P_{SoA} (`prmat` in code) of size $n_P \times h \times w$) that stores the feature crossed-products. In the *AoS* data layout these cubes are transformed into two 2D-arrays F_{AoS} and P_{AoS} of size $h \times (w \cdot n_F)$ and $h \times (w \cdot n_P)$. **Listing 6** is useful to illustrate the differences between both layouts by displaying the C macros used to navigate through these arrays in both implementations, it is worth to mention that the spatial coordinates x and y are handled in code by the indexes `j` and `i` and that the index `k` is used to select a particular feature.

The *SoA*→AoS transform consists in swapping the loop nests and changing the addressing computations from a 3D-form $cube[k][i][j]$ into a 2D-form like $matrix[i][j \times n + k]$, where n is the structure cardinal (here n_F or n_P). The lack of spatial locality within the features in the *SoA* representation is illustrated in **Figure 5-12**. In the *AoS* representation features features are gathered together in contiguous memory addresses.

```
//Feature images
FMAT(i,j,k) fmat[k][i][j]
//Crossed feature-products
PRMAT(i,j,k) prmat[k][i][j]
```

```
//Feature images
FMAT(i,j,k) fmat[i][(j)*NF+k]
//Crossed feature-products
PRMAT(i,j,k) prmat[i][(j)*NP+k]
```

Listing 6: C macros used to address the *SoA* cubes (on the left) or the *AoS* 2D-arrays (on the right) that store the set of features (and their crossed-products) used to compute the covariance matrix, `k` denotes the index of the feature (or feature-product) while `i` and `j` denote the image coordinates. `NF` and `NP` correspond to n_F and n_P respectively.

The product of features and its transformation are described in **Algorithms 16** and **17**. Due to commutativity of the multiplication, only half of the products have to be computed so, the loop on k_2 starts at k_1 (**Algorithm 16** line 3). As the two last stages are similar, only the generic versions of integral image computation are presented (the same for features and their crossed-products). The *SoA* version is shown in **Algorithm 18** while its *AoS* equivalent in **Algorithm 19**.

Table 5.4 compares the profiling results obtained for both versions, the gain of the *SoA*→*AoS* transformation is a speedup of about 1.5x. In turn, **Table 5.5** enlists some of the hardware events reported by Intel VTune Amplifier XE 2013 related to memory accesses (for instructions and data) in both *SoA* and *AoS* versions, the number cache misses and stalls is significantly reduced, for example the number of `MEM_LOAD_RETIRED.LLC_MISS` events which corresponds to last level cache misses and accesses to the local DRAM (signaled by the `MEM_UNCORE_RETIRED.LOCAL_DRAM`) is reduced almost by a half.

Table 5.4: *SoA* vs *AoS* profiling (full image version).

Function name	**Description**	*SoA* full-image		*AoS* full-image	
		CPU Time	%	CPU Time	%
`sumMatrices_SOA/AOS`	Computes I_F and I_P	25.616s	87.51%	16.665s	81.89%
`lbp_r1u8_operator_f32`	LBP operator	2.910s	9.94%	2.688s	13.21%
`rgb8matrix_get_L_RGB_F32`	Gets L,R,G and B features	0.504s	1.72%	0.469s	2.31%
`bilinearInterpol`	Bilinear interpolation	0.243s	0.83%	0.528s	2.59%

Table 5.5: SoA vs *AoS* hardware event count

Hardware Event Type	*SoA*	*AoS*		*AoS+SIMD*	
	Count	Count	Ratio (*vs. SoA*)	Count	Ratio (*vs. SoA*)
`MEM_LOAD_RETIRED.L1D_HIT`	41.418×10^9	35.828×10^9	0.865	12.272×10^9	0.296
`MEM_LOAD_RETIRED.L2_HIT`	10.402×10^6	5.214×10^6	0.501	33.821×10^6	3.251
`MEM_LOAD_RETIRED.LLC_MISS`	13.267×10^6	7.284×10^6	0.549	8.453×10^6	0.637

Algorithm 16: Products of features *SoA* version

```
1 k ← 0
2 foreach k1 ∈ [0..nF − 1] do
3     foreach k2 ∈ [k1..nF − 1] do
4         foreach i ∈ [0..h − 1] do
5             foreach j ∈ [0..w − 1] do
6                 P[k][i][j] ← F[k1][i][j] × F[k2][i][j]
7                 k ← k + 1
```

Algorithm 17: Products of features - *AoS* version

```
1 foreach i ∈ [0..h − 1] do
2     foreach j ∈ [0..w − 1] do
3         k ← 0
4         foreach k1 ∈ [0..nF − 1] do
5             foreach k2 ∈ [k1..nF − 1] do
6                 P[i][j × nP + k] ← F[i][j × nF + k1] × F[i][j × nF + k2]
7                 k ← k + 1
```

Algorithm 18: Integral image - *SoA* version, $n \in \{n_F, n_P\}$

Data: An image cube X of size $n \times h \times w$.
Result: An integral image cube I of size $n \times h \times w$.

```
foreach k ∈ [0..n − 1] do
    foreach i ∈ [0..h − 1] do
        foreach j ∈ [0..w − 1] do
            I[k][i][j] ← X[k][i][j] + I[k][i][j − 1] + I[k][i − 1][j] − I[k][i − 1][j − 1]
```

Algorithm 19: Integral image - *AoS* version, $n \in \{n_F, n_P\}$

Data: An image X of size $h \times (n \cdot w)$.
Result: An integral image I of size $h \times (n \cdot w)$.

```
foreach i ∈ [0..h − 1] do
    foreach j ∈ [0..w − 1] do
        foreach k ∈ [0..n − 1] do
            I[i][j × n + k] ← X[i][j × n + k] + I[i][(j − 1) × n + k] + I[i − 1][j × n + k] − I[k][i − 1][(j − 1) × n + k]
```

5.2.5 Code vectorization

Now that the features are laid out continuously in memory using the proposed *AoS* form, it is easier to extract parallelism using SIMD instructions. The final objective is to reduce the total number of memory accesses and arithmetic instructions. The most consuming part in the scalar implementation corresponds to the computation of the crossed feature products. The two internal loops in **Algorithm** 17 k_1 and k_2 are fully unrolled and the complete set of multiplications is obtained by first constructing the list of factors (in vector form) through permutation instructions (*e.g.*, `_mm_shuffle_ps` in SSE). For example, for a typical value of $n_F = 7$ features there are $n_P = 28$ final products, thus, seven vector products are enough (using four-element vectors). The associated vectors to obtain all these products in our example are (the numbers are the feature indexes):

$$
\begin{aligned}
[P_0, P_1, P_2, P_3] &= [F_0, F_0, F_0, F_0] \times [F_0, F_1, F_2, F_3]\,, \\
[P_4, P_5, P_6, P_7] &= [F_0, F_0, F_0, F_1] \times [F_4, F_5, F_6, F_1]\,, \\
[P_8, P_9, P_{10}, P_{11}] &= [F_1, F_1, F_1, F_1] \times [F_2, F_3, F_4, F_5]\,, \\
[P_{12}, P_{13}, P_{14}, P_{15}] &= [F_1, F_2, F_2, F_2] \times [F_6, F_2, F_3, F_4]\,, \\
[P_{16}, P_{17}, P_{18}, P_{19}] &= [F_2, F_2, F_3, F_3] \times [F_5, F_6, F_3, F_4]\,, \\
[P_{20}, P_{21}, P_{22}, P_{23}] &= [F_3, F_3, F_4, F_4] \times [F_5, F_6, F_4, F_5]\,, \\
[P_{24}, P_{25}, P_{26}, P_{27}] &= [F_4, F_5, F_5, F_6] \times [F_6, F_5, F_6, F_6]\,.
\end{aligned}
$$

In the mentioned case, all the seven features (contained in a pair of 128-bit SIMD registers) are distributed into fourteen vector combinations that are necessary to calculate the seven vector products. A collection of permutation intrinsics (SSE or Neon) is employed to get them. The initial 4-element vector pair `x0` and `x1`, they contain all the $n_F = 7$ features that are distributed to create the product factors (`a0` to `a6` and `b0` to `b6`) used to calculate the complete set of products. After this, seven SIMD instructions are then used to perform all the 28 multiplications of the original scalar algorithms. The whole

process is depicted in **Figure** 5-13. Two 128-width registers allows us to represent eight 32-bit integers or single precision values, for the case of a feature vector composed by $n_F = 7$ features (composed by $F_0, F_1, \cdots, F_6$) an eight feature (F_7) is appended as a padding element but it is not included in the feature combinations used to calculate the crossed products.

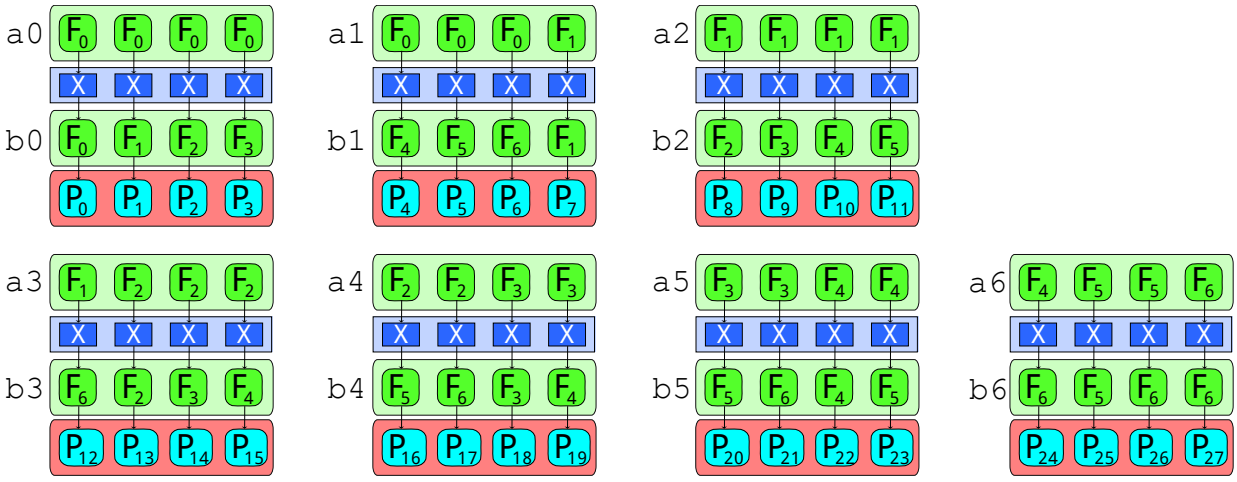


Figure 5-13: Seven vector products are enough to calculate the 28 crossed products for the $n_F = 7$ case with a degree of parallelism of 4 (four 32-bit integer values or single precision floating point values) with 128-bit width SIMD registers.

Seven vectors contain the 28 feature products and the 7th vector is 100% filled, but it will become sub-optimal if n_P is not divisible by the cardinal of the SIMD register (4 with SSE and Neon). For SSE some of the factor permutations can be achieved using only one instruction, the others need a maximum of two instructions, and because some permutations can be re-used to perform other permutations, it is possible to achieve a factorization over all the required permutations. In the case of our example ($n_F = 7$), only fifteen shuffles are required. In Neon things are more complex, and while some of the permutations can be done using 128-bits registers (giving us a degree of parallelism of 4), other permutations require instructions only available with 64-bit registers, like the *look-up table* instruction named `vtbl`. So in Neon the process is more complicated and 128-bit float registers should be: 1) split into 64-bit registers, 2) type-casted into 64-bit integer registers, 3) permuted with `vtbl` instructions 4) type-casted into 64-bit float registers and 5) combined into 128-bit float registers. In total 48 SIMD Neon instructions are required to create the seven pairs of products.

Tables 5.6 and **5.7** provide an estimation of the algorithmic complexity and the amount of memory accesses for both scalar and SIMD (SSE and Neon) versions. They also provide the arithmetic intensity (AI) (a measure popularized by Nvidia) that compares the number of arithmetic operations (including the number of permutations for SIMD version) and the number of memory accesses. Note that the scalar version has a low AI of 0.5 as the number of memory accesses is twice the number of operations. It can also be noticed that the SIMD version has $\times 2.7$ less operations for SSE (respectively 1.6 for Neon) and $\times 4.8$ less memory accesses, thus the AI ratios reach 0.9 for SSE and 1.5 for Neon.

A comparison of the profiling results obtained using *AoS* and *AoS+SIMD* versions of the algorithm is shown in **Table 5.8**. The part of the algorithm which was vectorized is represented in code by the function `sumMatrices_AOS_SIMD` and has a speedup of 2.61x with respect to its scalar version making it about four times faster than the *SoA* baseline version. For the scalar *SoA* baseline version, this function represented about 84.07% of the running time execution, in the scalar *AoS* version it was reduced to about 81.89%. With the *AoS+SIMD* version it represents now only 64.45%. The total execution

Table 5.6: Complexity and arithmetic intensity of scalar *AoS* version.

AoS scalar version with 3 loops

Instructions	**Arithmetic instructions**		**Memory accesses**		**Arithmetic Intensity**
	MUL	ADD	LOAD	STORE	
Product of features	n_P	0	$2n_P$	n_P	-
Integral of features	0	$3n_F$	$4n_F$	n_F	-
Integral of products	0	$3n_P$	$4n_P$	n_P	-
Total	n_P	$3(n_P + n_F)$	$6n_P + 4n_F$	$2n_P + n_F$	-
With $n_P = n_F(n_F+1)/2$	$2n_F^2 + 5n_F$		$4n_F^2 + 9n_F$		-
and $n_F = 7$	133		259		0.5

Table 5.7: Complexity and arithmetic intensity of the SIMD *AoS* version.

AoS SIMD (with $n_F = 7$) version with 3 loops

Instructions	**Arithmetic instructions**		**Memory accesses**		**Arithmetic Intensity**
	MUL	ADD	LOAD	STORE	
Product of features	7	0	2	7	-
Integral of features	0	21	28	7	-
Integral of products	0	6	2	2	-
Total SSE (+ 15 PERM)	49		54		0.9
Total Neon (+ 48 PERM)	82		54		1.5

time of *AoS+SIMD* for the Pedxing3 sequence on a Nehalem architecture (Intel Core i5) was about 11.525 the *AoS+SIMD* version is 2.78x faster than the *SoA* one and 1.82x faster than the scalar *AoS*.

Table 5.8: *AoS* and *AoS+SIMD* profiling results (full image version).

Function name	**Description**	*AoS* full-image		*AoS+SIMD* full-image	
		CPU Time	%	CPU Time	%
`sumMatrices_AOS/AOS_SIMD`	Computes I_F and I_P	16.665s	84.07%	6.382s	64.46%
`lbp_r1u8_operator_f32`	LBP operator	2.688s	13.56%	3.001	30.31%
`rgb8matrix_get_L_RGB_F32`	Gets L,R,G and B features	0.469s	2.37%	0.518s	5.23%

5.2.6 Multi-thread implementation

Concerning our problem, two different strategies are possible to parallelize the covariance descriptor and tracking algorithm. One scheme would be to implement a task-level parallelism using OpenMP or Pthreads (POSIX Threads) (to create a pool of threads and distribute the set of targets among them). The second strategy uses loop-level parallelism: the operations on the images are divided into balanced groups of rows per thread. This approach is the more straightforward to apply using OpenMP since (as we will see next) it only demands to add some pragma lines just before the most consuming loops in our source code, this last approach is evaluated here.

Figures 5-14, **5-15** and **5-16** show the evaluation results of three different parallelized versions of the covariance

tracking algorithm: the *SoA* version with OpenMP (*SOA+OpenMP*) versus the SIMDized *AoS* version *AoS+SIMD*. Indeed, for a common 4-core General Purpose Processor (GPP) the degree of parallelism with a multi-threaded version and the SIMDized version is the same (*i.e.* four). Results are provided graphically and in cycles per point (*cpp*) which corresponds to the number of cycles invested by pixel

$$\text{total}cpp = \frac{\text{cpu cycles}}{N^2}, \tag{5.8}$$

where N^2 corresponds to an image of size $N \times N$. The *cpp* results are plotted *versus* the number of pixels (image size). Detecting the *cache overflow* (when data do not fit in the cache) is possible using the *cpp* metric, it occurs when the curve increases significantly. This metric reflects the influence of memory transfers on computation using a fair metric which is independent from the processors clock and which offers a valid methodology to compare different architectures.

Three different versions were tested (*SoA+OpenMP*, *AoS*, *AoS+SIMD*), they were benchmarked on three generations of Intel processors: Penryn4, Nehalem and SandyBridge and varying the images sizes from 128×128 up to 1024×1024. As it appears in **Figures 5-14** and **5-16** the 4-threaded version is always slower than a 1-threaded SIMD version, eight threads are required on the Nehalem (**Figure 5-15**) to be faster. The reason is the low arithmetic intensity that induces a high stress on the architecture's buses and also because the *SoA* version requires $n_P = 28$ active references in the cache, that is more than the usual L2 or L3 associativity (24 on the Intel processor).

The impact of the *AoS+SIMD* optimizations on the Penryn architecture (**Figure 5-14**) is significant in comparison to the other two versions. It improves from around 250 *cpp* to just 125 *cpp* which corresponds to a speedup of about 2x. The problem is that this improvement is limited to images sizes of about 256×256 or less, if images are bigger, then the performances offered by *AoS+SIMD* and *SoA+OpenMP4* are very similar (275 *cpp*). The *AoS+SIMD* version demands 2.2x more cycles for each pixel after the cache overflow! That increase on the cycles per pixel suggests that the problem here is memory bounded and not limited by the number of arithmetic computations.

For the Nehalem architecture things are much more better, this is visualized in **Figure 5-15** where the *SoA+OpenMP8* version is here the fastest one, it requires just 50 *cpp* for images smaller than 400×400, for bigger images the cache overflow effect is noticeable but the transition rate is very smooth and goes up just to about 70 *cpp*. The effect of cache overflow here is less dramatic (just 1.4x more instructions per cycle instead of 2.2x for the Penryn4). The *AoS+SIMD* version offers a comparable performance: it demands about 65 *cpp* before the cache overflow (for images bigger than 256×256), the transition period is a little bit sharper here and stabilizes around 90 *cpp* for image sizes bigger than 400×400. In general, this algorithm requires 1.3x more cycles than the multi-threaded *SoA+OpenMP8 version*. The scalar *AoS* version remains very far demanding $\times 2.8$ more *cpp* for small image sizes of about 400×400 and $\times 2.28$ *cpp* for bigger images.

Finally, **Figure 5-16** shows the results for the SandyBridge architecture, the reduced number of cores in comparison to the previous architecture is noticed here and the vectorized *AoS+SIMD* version is again the fastest one. It demands about 65 *cpp* before the cache overflow transition period which starts for images bigger than 256×256 and transitions relatively sharply up to 110 *cpp* where it stabilizes for bigger images than 400×400 (an increase of $\times 1.69$). The scalar *AoS* and the multi-threaded **SoA+OpenMP4** remain far behind demanding $\times 2.30$ more *cpp* than the *AoS+SIMD version* for small images (smaller than 256×256) and $times 1.63$ more *cpp*'s for bigger images. The effect of the cache overflow is smoother for the multi-threaded *SoA+OpenMP4* version, but it demands about $\times 1.45$ more *cpp*'s than the vectorized *AoS+SIMD* version.

From this experiments we can conclude that although the effect of cache overflow is less dramatic for the multi-threaded *SoA+OpenMP* version (which can be attributed to the additional cache memory spaces are used when the data is distributed among the unshared core cache memories), the real improvement to the covariance descriptor computation algorithm was obtained by the *SoA→AoS* transformation combined with the code vectorization. For the rest of the chapter, the vectorization of code is the only architectural optimization considered as realistic for this algorithm.

In the next subsection, the last optimization to the covariance descriptor computation algorithm is given. It is a loop fusion transformation combined with other scalarization techniques to keep the intermediate and re-usable calculations in local registers to reduce the number of memory accesses loads and stores and increase the arithmetic intensity.

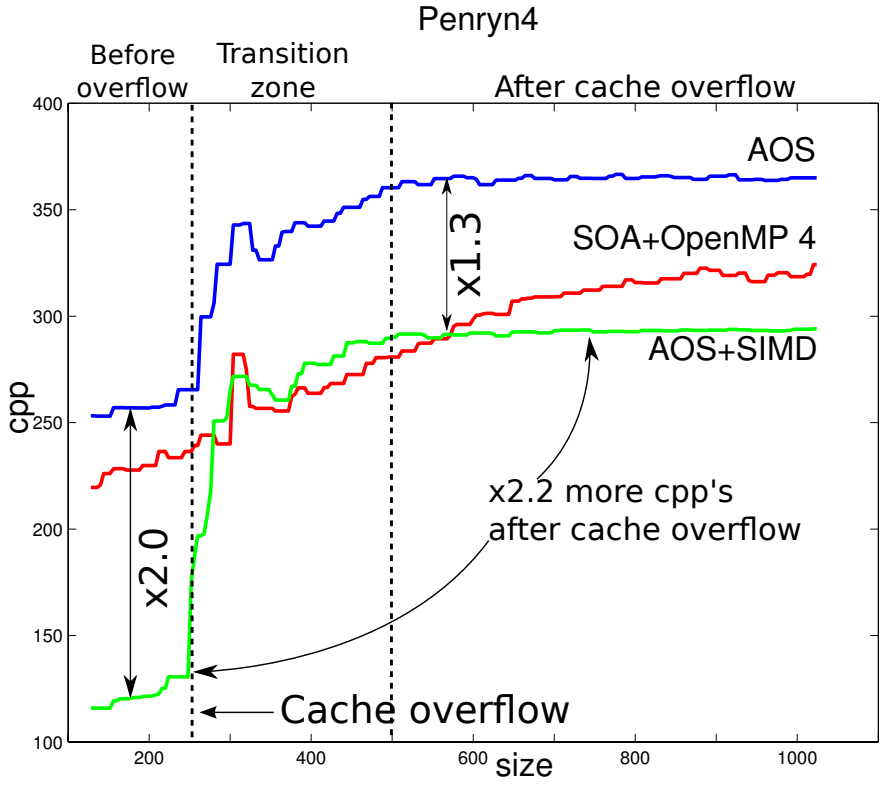


Figure 5-14: Performance in *cpp* of a 1×4-core Penryn for image sizes $\in [128..1024]$. Before the cache overflow transition starts (close to 250×250), the *AoS+SIMD* is $\times 2$ faster than the *SoA* version, once the transition ends (at 500×500), the performance drops significantly and the algorithm demands $\times 2.2$ more *cpp*'s than before the cache overflow, the speedup in this zone is just $\times 1.3$.

5.2.7 Loop fusion and scalarization

loop-fusion and serialization transformations is used here to increase the arithmetic intensity (AI) ratio by reducing the total amount of communication expressed in the form of loads and stores. Three *AoS* algorithm blocks are fused together into a single one:

- the first block is an instance of **Algorithm** 17 which computes from the n_F first-order features the set of n_P crossed-feature products (second order features),

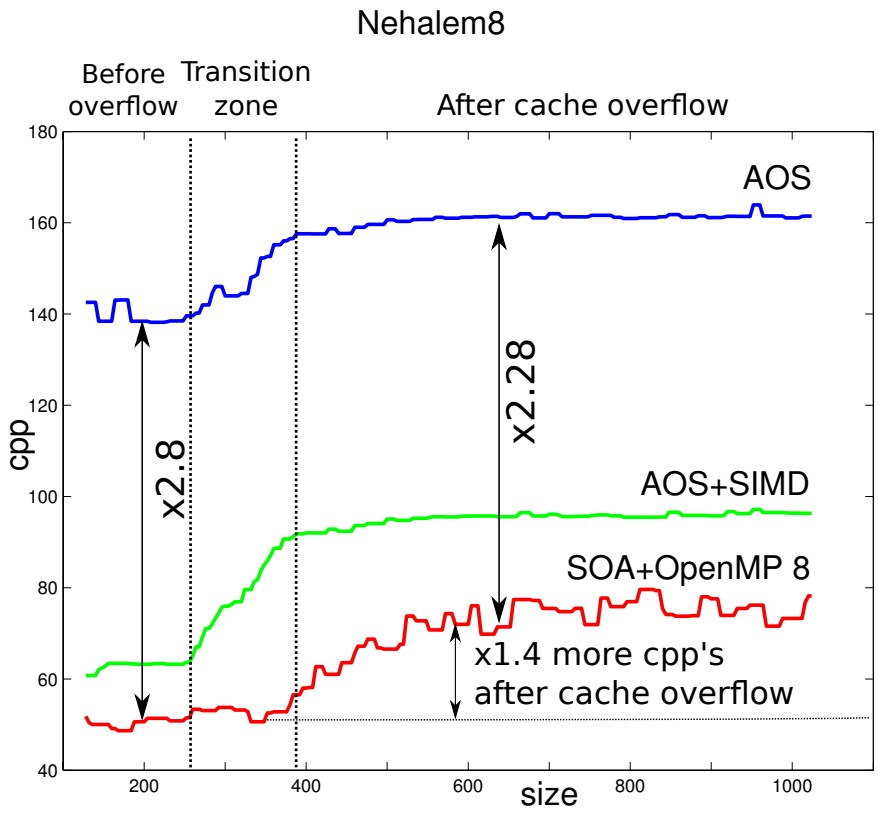


Figure 5-15: Performance in *cpp* of a 2×4-core Nehalem for image sizes $\in [128..1024]$. Contrary to the Penryn architecture, the *SoA+OpenMP* is faster on the Nehalem processor than the *AoS+SIMD* and the *AoS* versions thanks to the larger number of threads (8 *vs*. 4). Before the cache overflow starts at 250×250, the speedup provided by OpenMP is $\times2.8$ faster than the scalar and single threaded *AoS*. After the overflow transition ends at 650×650 for the *SoA+OpenMP* version, the speedup obtained drops to $\times2.3$ because the algorithm in this regions demands $\times1.4$ more *cpp*'s.

- the second algorithm is the first order instantiation of **Algorithm** 19 which computes the first order n_F feature integral images,
- and finally, a second instance of **Algorithm** 19 is used to compute the n_P second-order integral images.

Three different versions of the loop-fusion together with serialization transformations were tested, the first one is a scalar parametric version (for variable n_F) that fuses the external `i`-loops and keeps the three `j`-loops unchanged. The second one is a specialized version for the particular case of $n_F = 7$ where the three internal loops are fused together too. The third one is the SIMDized version of the second one. The internal loop fusion allows to save in LOAD/STORE instructions in order to avoid writing the products of features into memory and to read them afterwards to compute the integral image of products.

In this particular case, loop-fusion and serialization transformations has been done by hand, but some tools like PIPS [96] can do such kind of transformations automatically [67]. The expected complexity of the transformations is reported in **Table 5.10** for the scalar version, **Table 5.11** reports the expected complexity for the vectorized SSE and Neon versions.

The transformed versions of the *AoS* and *AoS+SIMD* are from now on referred as *AoS* and *AoS+T+SIMD* respectively. These series of transformation are the last optimization provided in this thesis to accelerate the execution of the covariance descriptor computation algorithm. As these versions of the algorithm were ported to the embedded platforms Intel U9300

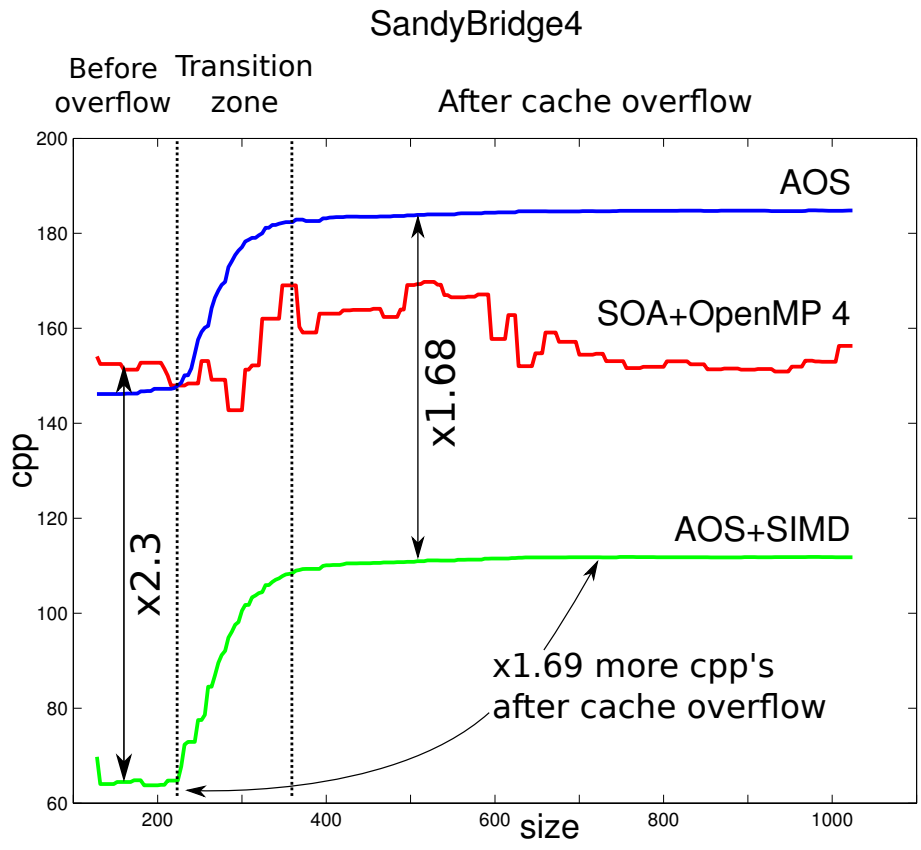


Figure 5-16: Performance in *cpp* of a 1×4-core SandyBridge4 for image sizes $\in [128..1024]$. As the available number of threads for this processor is only 4, the *AoS+SIMD* is again faster than *SoA+OpenMP 4* (similarly to the case of the Penryn architecture in **Figure 5-14**). Before the cache overflow transition begins (at 225×225), the speedup obtained is $\times 2.3$. After the transitions ends (before 400×400), the speedup drops to $\times 1.68$ as the algorithm demands for this region $\times 1.69$ more *cpp*'s.

and ARM Cortex-A9 benchmark results for this series of transformation are provided in the following subsection.

Table 5.10: Complexity and arithmetic intensity for the loop fusion + scalarization version.

AoS scalar version + Loop Fusion

Instructions	**Arithmetic instructions**		**Memory accesses**		**Arithmetic Intensity**
	MUL	ADD	LOAD	STORE	
Integral of features	0	$2n_F$	$2n_F$	n_F	-
Integral product of features	n_P	$2n_P$	n_P	n_P	-
Total	n_P	$2(n_P + n_F)$	$n_P + 2n_F$	$n_P + n_F$	-
Total with $n_P = n_F(n_F + 1)/2$	$1.5n_F^2 + 3.5n_F$		$n_F^2 + 4n_F$		-
Total with $n_F = 7$	98		77		1.3

Table 5.11: Complexity and arithmetic intensity of scalar and SIMD versions with loop-fusion and serialization transformations and scalarization.

AoS SIMD (with $n_F = 7$) version + (loop fusion + scalarizations)

Instructions	**Arithmetic instructions**		**Memory accesses**		**Arithmetic Intensity**
	MUL	ADD	LOAD	STORE	
Integral of features	0	4	4	2	-
Integral product of features	7	14	7	7	-
Total SSE (+ 15 PERM)	40		20		2.0
Total Neon (+ 48 PERM)	73		20		3.7

5.2.8 Implementation on embedded systems

In this subsection the implementation results on *embedded* processors like the Intel ULV (Ultra Low Voltage) Penryn U9300 (that belongs to the Penryn family) and the ARM Cortex-A9 are provided. The average power consumption (TDP) of these processors is very low: about 10 W and 1 W respectively. Both run at very close frequencies (1.2 GHz and 1.0 Ghz) but there are significant differences. For example, the Intel ULV U9300 uses out of order execution to reduce the impact of costly delays while ARM Cortex-A9 does in order execution which means that it follows the strict order of the program assembly instructions and does not tries to execute independent instructions while some data is retrieved. Another important difference is that the SSE4.1 offers a throughput of one SIMD operation per cycle while all Neon instructions on the Cortex-A9 take two cycles.

Figures 5-17 and **5-18** provide the measured *cpp*'s for the *SoA*, *AoS*, *AoS+SIMD*, *AoS+T* and *AoS+T+SIMD* versions of the algorithm implemented on the Intel U9300 and ARM Cortex-A9. In both processors, the *SoA* version is very inefficient specially when it is compared with the more efficient one (*i.e.* AoS+T+SIMD). But there are two big differences between them, the first one is impact of the optimizations: for the Cortex-A9 when images larger than 100×100 are applied the only possibility for optimization is to apply the loop-fusion and serialization transformations, the impact of the Neon instructions is insignificant and the number of *cpp*'s required by the *AoS+SIMD* version is very similar to those required by the *AoS* version. The same occurs for the case of the *AoS+T+SIMD* and *AoS+T* versions, where it is not clear if Neon reduces the execution time. The problem comes from the memory hierarchy of the Cortex-A9 and from the number of extra instructions required by Neon to perform the permutations (48 for Neon *versus* only 15 for SSE). The second difference is the *cpp* range of values between both architectures: the Intel ULV 9300 demands about $\times 4.5$ less *cpp*'s than ARM Cortex-9 an important

factor here is the latency of the instructions.

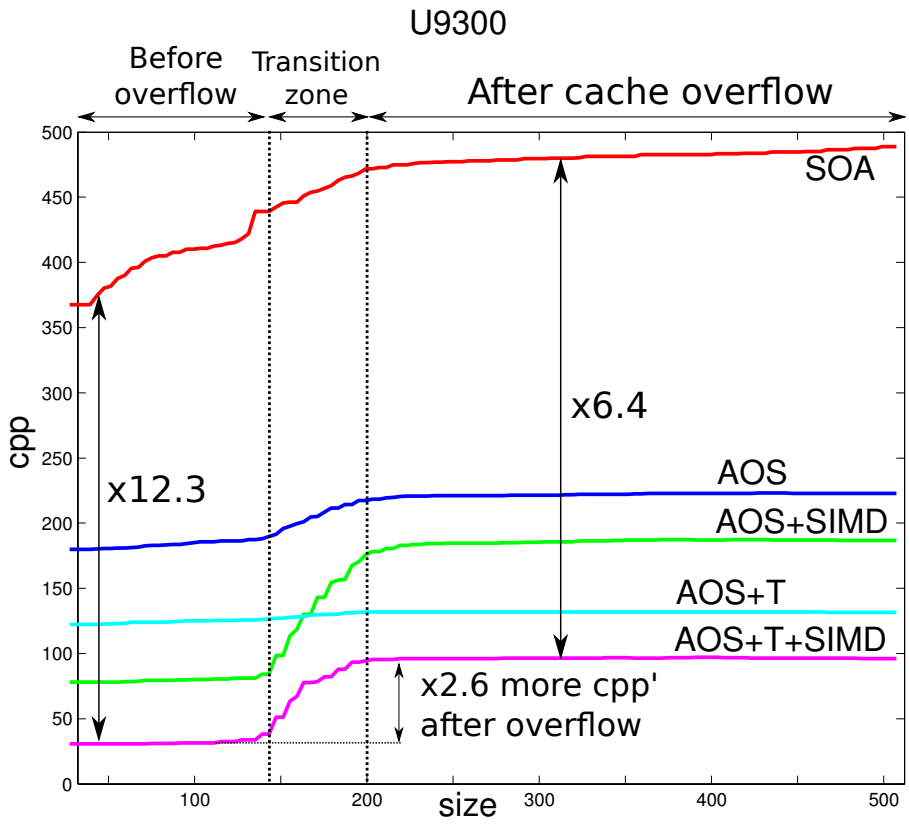


Figure 5-17: Performance in *cpp* for the Intel ULV U9300 for image sizes $\in [32..512]$. Before the cache overflow transition starts (close to 150×150 image sizes), the *AoS+T+SIMD* version is $\times 12.3$ faster than the *SoA*. After the cache overflow transitions ends (at 200×200 image sizes), the speedup goes down to $\times 6.4$ because the *AoS+T+SIMD* requires $\times 2.6$ more cpp's after the overflow.

Table 5.12: Impact of loop-fusion and serialization transformations, speedups obtained for Intel's ULV U9300 and Cortex-A9.

Algorithm version	Intel U9300	ARM Cortex-A9
AoS / *AoS*+T	$\times 1.8$	$\times 3.3$
AoS+SIMD / *AoS*+T+SIMD	$\times 2.2$	$\times 3.2$
SoA / *AoS*+T+SIMD	$\times$**6.4**	$\times$**3.4**

Let us now focus on the impact of the loop-fusion and serialization transformations, (**Table** 5.12) lists the speedups obtained using the different versions of the algorithm in both architectures. The speedup of the *AoS+T* compared to the *AoS* version is about $\times 2$ for Intel and $\times 3.3$ for ARM, very similar factors are obtained when comparing the pair of SIMD versions *AoS+SIMD* and *AoS+T+SIMD* in both architectures, speedups of about $\times 2.2$ and $\times 3.2$ were obtained. Finally, comparing to the baseline *SoA* version, speedups of about $\times 5.3$ and $\times 3.4$ were obtained for each architecture. Loop-fusion and serialization optimizations are mandatory for this algorithm, improved performances are obtained creating specialized cases for different values of n_F and applying loop-unwinding.

Concerning power efficiency, there is a factor 4 of speed between the Intel U9300 and the Cortex-A9. However, as the

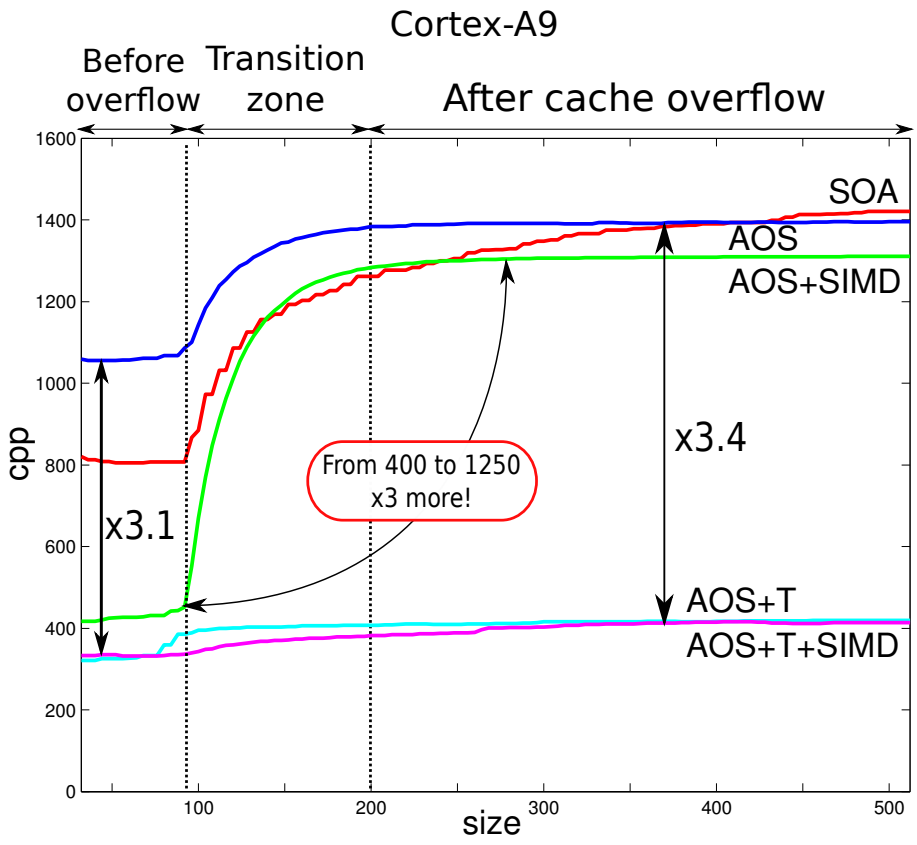


Figure 5-18: Performance in *cpp* for the Cortex A9 processor for image sizes $\in [32..512]$. The best performance is offered by the *AoS+T+SIMD* version of the algorithm. Before the cache overflow transition starts (close to 100×100 image sizes), the *AoS+T+SIMD* version is $\times 3.1$ faster than the *AoS*. After the cache overflow transitions ends (at 200×200 image sizes), the speedup goes up to $\times 3.4$ because the *AoS+T+SIMD* is not affected by the cache overflow and requires nearly the same number of *cpp*'s than before the overflow. Before the transition, the *AoS+SIMD* version is nearly as fast as the *AoS+T* and *AoS+T+SIMD* but after the overflow its performance drops and demands $\times 3$ more *cpp*'s.

power consumption (TDP) of the Cortex is approximatively 10 times less than that of the U9300, the Cortex-A9 is about two times more power efficient than the U9300.

Now that the covariance descriptor computation algorithm has been optimized, let us now focus on the impact of those optimization in a tracking application based on the covariance descriptor. The same group of sequences used for the baseline evaluation (Panda and Pedxing) were evaluated here. For both of them, the execution times are given in *cpp* and reporting each version of the algorithm. The *SoA* version is the baseline version, *AoS++* stands for the *AoS+T+SIMD* one. Two counter-intuitive results can be noticed. The first one is the features computation *cpp*: it is lower for *SoA*. The reason is obviously the memory layout of *SoA* (versus *AoS*) when computing the features and storing them into a cube or a matrix, this problem can be solved using an integrated features computation function instead of calculating each feature in a separate function (this solution is left for future work). The second counter-intuitive result is the huge difference on the number of cycles per pixel invested for tracking on each sequence, this is justified by the fact that this function does not depend on the size of the image neither on the size of the targets but on the number of features used to construct the covariance matrices. As this value is normalized by the number of pixels it is less for the *Pedxing3* sequence which has about 4 times more pixels.

Concerning the acceleration, we can see (**Tables 5.13** and **5.14**) that the optimization of the kernel provides a speedup of $\times 2.9$ for Intel and $\times 2.2$ for ARM that assets the need of all optimizations. As both processors have two cores, all the processing parts can be done either on one core (the execution time is the sum of all parts) or on two cores (the biggest part is on one core and the two other parts are on the second core). With such a coarse grain thread distribution, the Intel U9300 can track targets in real-time for 640×480 images, while the ARM Cortex-A9 can do it for image sizes up to 320×240. As said previously, there is a factor 10 for power consumption, and only a factor 4 in execution time. So for small images (up to 320×240) the Cortex-A9 is the best choice as it is real-time and more power efficient than the Intel. While for bigger sizes (up to 640×480), the Intel is the only choice for real-time implementation.

Before the optimizations, the kernel function of the covariance descriptor computation represented about 80% of the execution time on the Intel ULV 9300 processor. In the final *AoS+T+SIMD* version it represents 35% and the features computation algorithm represents now about 60% of the execution time on this platform. With respect to the ARM Cortex-A9, before the optimizations, the kernel represented about 73% of the total execution time, after the *AoS+T+SIMD* optimizations it only represents 45%. In both platforms, the most consuming part after the optimizations corresponds to the features computation algorithm, it is very likely that there is still plenty of room for more optimizations in this part, particularly in the integration of the features computation and the data structures used by the covariance descriptor computation algorithm.

Table 5.13: *cpp* and execution time for Intel U9300

Sequence	Panda (312×233)		Pedxing (640×480)	
	SoA	*AoS++*	*SoA*	*AoS++*
Features computation (*cpp*)	128	150	128	150
Kernel computation (*cpp*)	599	87	618	91
Tracking (*cpp*)	23	23	11	11
Total (*cpp*)	738	248	769	264
Kernel / total	81 %	35 %	80 %	34 %
Total speed-up	$\times 2.9$		$\times 2.8$	
1-C execution time (ms)	45	15	197	68
2-C execution time (ms)	36	9	158	38

Table 5.14: *cpp* and execution time for ARM Cortex-A9

Sequence	Panda (312×233)		Pedxing (640×480)	
	SoA	*AoS++*	*SoA*	*AoS++*
Features computation (*cpp*)	461	461	486	486
Kernel computation (*cpp*)	1491	395	1600	415
Tracking (*cpp*)	96	96	19	19
Total (*cpp*)	2048	952	2106	921
Kernel / total	73 %	42 %	73 %	45 %
Total speedup	$\times 2.2$		$\times 2.2$	
1-C execution time (ms)	149	69	647	283
2-C execution time (ms)	108	36	492	149

5.3 Conclusions

This chapter has presented a real-time implementation of the robust ELBCM-based covariance tracking algorithm developed in the previous chapters. The covariance descriptor algorithm has a parameterizable complexity that can be adapted to the number and nature of features for trade-off between robustness and execution time. Classical software and hardware optimizations have been applied: SIMDization and loop-fusion and serialization transformations combined with the *AoS→SoA* transform to accelerate the kernel of the algorithm. These optimizations allow a real-time execution on *heavy* embedded systems like Intel U9300 and on *light* ones like the ARM Cortex-A9. The loop-fusion technique accompanied with other scalarization techniques represents provides the more important gains of performance, it is possible to use SIMD instructions together with this technique, the *AoS+T+SIMD* technique requires gets a speedup of about $\times 2$ mixing these techniques, unfortunately, this doesn't work so well on the ARM Cortex-9 platform where the improvement on the performance is minimal. This algorithm will be ported soon to other platforms such as the ARM Cortex-15 and more other more energy-efficient Intel processors. As far as I know, this implementation of the covariance tracking algorithm is the first real-time implementation for embedded systems that keeps all its qualities and improves its robustness.

General conclusions and perspectives

This thesis has described our work in image processing and computer architecture. The aim was to propose methodologies for object re-identification and tracking with their implementation on multi-core embedded systems. This work has lead to several publications but also to a a real integration in the European project *Surveillance imProved sYstem* of ITEA2, coordinated by Cassidian (EADS).

Starting from a general overview on different features available at a pixel-level, the first chapter has introduced some of the possible object representations. Due to the large amount of methods, it was difficult to make a decision. Indeed, the representation has to be invariant against distortions such as illumination changes, noise or geometric changes while being well separated from the other objects of the scene and from the background.

The second chapter has presented a few algorithms commonly used for object detection and matching for re-identification and tracking. When only the object family to be detected is known (pedestrian, car, plane...), learning-based methods are commonly used. For re-identification and tracking, it is assumed that some information are available on a specific object of interest and that this object has to be retrieved in another context or in another frame of the same video. Concerning visual tracking, it is complicated to find an unified method which could be efficient whatever the nature of the object and whatever the context. In our work, we were more interested in three different methods:

- Flock of Trackers (*FoT*) which represents the object as a set of local areas which are tracked by KLT-like methods and robust statistics on the computed optical flow are used to provide the global motion vector. This method performs well but only for objects of rigid motion, which have enough texture.
- Mean-Shift tracking (*MS*) which represents the object by its color distribution and finds its new location in the video by optimization. This method is interesting because the distribution can be computed for any object, contrary to feature points.
- Covariance tracking (*CT*) which represents the object by a covariance matrix of feature vectors.

Our choice has been to study some color and texture features and to evaluate them in terms of quality, *i.e.* color invariance and precision, and computation times. Then, as an object representation, we have finally chosen covariance-based techniques for several reasons:

- it provides an unified representation for object detection and matching;
- it is compact with a fix size whatever the object size;
- it mixes heterogeneous information such as spatial, color and texture what makes it highly discriminant.

Contributions

The first group of contributions presented in **Chapter 3** reinforce the adaptability of the tracking algorithm to the context.

First, the context can change in terms of illumination. To tackle this problem, we proposed a method which selects color features automatically according to their relevance. The color is used only when the saturation is high enough. In addition, a simple and compact color invariant has been proposed. This contribution has been published in [113] and has won the Best Student Paper award. The resulting invariant is used to improve the results of the *FoT* algorithm. It can also be used to form an *ELBCM*-based texture and color-invariant covariance descriptor (see the tests in **Section 4.2.2**).

Second, the chosen tracking algorithm can fail for some reason, for instance a loss of discriminant power or an occlusion. Additionally, the tracking algorithm can fail because it is not well chosen for a given object, for example, when *FoT* is used on an object which has no salient feature points. The *FoT*+*CT* and *MS*+*CT* switching mechanisms increase the robustness of the final tracking algorithm by compensating their individual failures this research work has lead to a journal publication [79].

Finally the aim of the last group of contributions to the field of computer vision was to improve the distinctiveness of the covariance descriptor by mixing texture (in the form of *LBP* angles) and color information. The resulting *ELBCM* descriptor was published in [115], it is presented in **Chapter 4**. *ELBCM* can be used for many applications such as facial expression and texture recognition, object matching and tracking, target re-identification, etc. Our work in target re-identification using an array of covariance matrices and evaluating different color-spaces led to a conference publication [114]. Mixing the target appearance and their dynamics it was possible to propose a multi-target tracking procedure based on the tracking-by-detection approach which is suitable for on-the-fly operation [116].

Chapter 5 analyzed the kernel of the covariance descriptor computation algorithm and proposed some simple techniques to accelerate its implementation on embedded *GPP* processors such as the ARM Cortex-A9. The set of acceleration techniques prosed were published in [117] and they are completely scalable for future processor architectures which provide a higher degree of parallelism (in the form of wider SIMD registers or a higher number of cores). This algorithm has already been ported to a faster embedded platform (ARM Cortex-A15) and our team at the LRI is now discussing with representatives of Intel and Kalray to create collaboration projects to port this algorithm to their many-core platforms: the Intel Xeon Phi co-processor and the MPPA-Manycore of Karlay.

Future work

There are still a lot of topics to explore in the domain of detection, matching and tracking using covariance matrices. The evaluation of our algorithms, particularly the proposed *ELBCM* method, can be continued. It could be interesting to test the training of an *ELBCM*-based pedestrian detection using Kernel methods on Riemannian manifolds (as described in **Section 2.1.3.3**). We would also like to test the *ELBCM* descriptor using the L^2ECM approach proposed in [82].

For the multi-target tracking algorithm, the plans for the future are based on the exploration of new techniques to use the appearance information (modeled by covariance matrices) and the dynamics of the targets using integer and linear programming as in [95]. To optimize the algorithm there are two main points to treat: to tackle precision for integral image computation for wider images (typically HD 1920×1080 pixels) (and how to parallelize it if it makes sense with the required precision); and the efficient parallelization of the multi-target tracking on multi-core and many-core systems like Intel XeonPhi for high-performance systems and Kalray MPPA for embedded systems.

APPENDIX A

KLT optical flow models

In this appendix two models used in the KLT optical flow are detailed, the translation model which is valid only when the surface is locally planar around the point and the motion is fronto-parallel. And the affine transformation model which handles more complex transformations rather than simple translations. Affine maps and homographies may associate different velocities to different points inside the window. However, the size of the window area has to be chosen as a trade-off, it has to be long enough to have enough information to estimate the motion model and small enough to avoid being time consuming.

A.1 KLT with a translational motion model

Dropping the time variable we have that $J(\mathbf{x}) = I(x, y, t + \Delta t)$ and $I(\mathbf{x} - \mathbf{d}) = I(x - \xi, y - \eta, t)$, the local image model is

$$J(\mathbf{x}) = I(\mathbf{x} - \mathbf{d}) + n(\mathbf{x}) \tag{A.1}$$

where $\mathbf{d}$ is the displacement vector which minimizes the double integral over the neighborhood window $\mathcal{W}$:

$$\epsilon = \int_{\mathcal{W}} \left[I(\mathbf{x} - \mathbf{d}) - J(\mathbf{x})\right]^2 w d\mathbf{x}. \tag{A.2}$$

In equation (A.2), w assigns a weight to each point inside $\mathcal{W}$. Usually, the value of w is determined with a Gaussian-like function centered at $\mathcal{W}$ so that pixels close to the borders of the region are considered less important than pixels close to the center. In the simplest case, a uniform value such as $w = 1$ is assigned to every point.

In the general case, there is no linear relation between pixel intensities. When the displacement vector $\mathbf{d}$ is small, a good linear approximation is obtained considering the Taylor expansion up to the first order term. The result of this linearization is

$$I(\mathbf{x} - \mathbf{d}) = I(\mathbf{x}) - \mathbf{g} \cdot \mathbf{d}, \tag{A.3}$$

where $\mathbf{g} = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$ is the spatial gradient defined in **Section 1.1.2**.

Substituting (A.3) into the residual error of (A.2) we get

$$\epsilon = \int_{\mathcal{W}} \left[I(\mathbf{x}) - \mathbf{g} \cdot \mathbf{d} - J(\mathbf{x})\right]^2 w d\mathbf{x} = \int_{\mathcal{W}} (h - \mathbf{g} \cdot \mathbf{d})^2 w d\mathbf{x}, \tag{A.4}$$

where $h = I(\mathbf{x}) - J(\mathbf{x})$.

Equation (A.4) is a quadratic function of the displacement $\mathbf{d}$. Differentiating it with respect to $\mathbf{d}$ and setting the result

to zero conduces us to the optimal value of **d**:

$$\int_{\mathcal{W}} (h - \mathbf{g} \cdot \mathbf{d})\mathbf{g} w d\mathbf{A} = 0. \tag{A.5}$$

Knowing that $(\mathbf{g} \cdot \mathbf{d})\mathbf{g} = (\mathbf{g}\mathbf{g}^T)\mathbf{d}$ and assuming **d** to be constant within $\mathcal{W}$, we have

$$\left(\int_{\mathcal{W}} \mathbf{g}\mathbf{g}^T w d\mathbf{A} \right) \mathbf{d} = \int_{\mathcal{W}} h\mathbf{g} w dA. \tag{A.6}$$

This system of equations can be rewritten as

$$\begin{gathered} G\mathbf{d} = \mathbf{e} \\ \underbrace{\left(\int_{\mathcal{W}} \mathbf{g}\mathbf{g}^T w d\mathbf{A} \right)}_{G} \mathbf{d} = \underbrace{\int_{\mathcal{W}} (I - J)\mathbf{g} w dA}_{\mathbf{e}} \end{gathered} \tag{A.7}$$

A.2 KLT with the affine motion model

Putting translation aside, more complex changes are modeled by the **affine transformation model**, represented by the following equation

$$I(x, y, t + \Delta t) = I\left(x - \xi(x, y, t, \Delta t), y - \eta(x, y, t, \Delta t)\right), \tag{A.8}$$

where a later image taken at $t + \Delta t$ is obtained by moving every point from the image at t by a suitable amount. This amount is $\delta = (\xi, \eta)$ and is called the displacement of the point $\mathbf{x} = (x, y)$.

When using the affine translation model, it is allowed to model different displacements co-existing inside the same window $\mathcal{W}$, as such, it makes little sense to speak of a general feature window displacement. The *affine motion field* is a better representation, here δ is

$$\delta = D\mathbf{x} + \mathbf{d} \tag{A.9}$$

where

$$D = \begin{bmatrix} d_{xx} & d_{xy} \\ d_{yx} & d_{yy} \end{bmatrix} \tag{A.10}$$

is a deformation matrix and **d** represents the translation vector of the feature window's center. If image coordinates **x** are measured with respect to the window's center, then, a point **x** in the first image I moves to $A\mathbf{x} + \mathbf{d}$, where $A = \mathbf{1} + D$, where **1** represents the 2×2 identify matrix. The transformation is thus expressed as

$$J(A\mathbf{x} + \mathbf{d}) = I(\mathbf{x}). \tag{A.11}$$

With the affine model, tracking means determining the six parameters of the affine transformation: four deformation parameters in matrix D and two translation parameters in vector **d**. The quality of the estimation depends on many factors such as: the size of the window, image texturedness and the amount of motion between the frames. Because deformation changes are more notable far from the window's center larger windows allows us to estimate the values of matrix D

with a higher degree of confidence than with small windows. However, smaller windows are preferable for displacement estimations as it is less probable for them to cover regions with depths discontinuities.

The problem of determining the motion parameters is that of finding A and $\mathbf{d}$ that minimize the *dissimilarity* (matching error)

$$\varepsilon = \int_{\mathcal{W}} [J(A\mathbf{x} + \mathbf{d}) - I(\mathbf{x})\,]^2\, w(\mathbf{x}) d\mathbf{x}. \tag{A.12}$$

Now, to minimize (A.12) it is necessary to differentiate it with respect the unknown entries of matrix D and vector $\mathbf{d}$ and set the result to zero. The resulting system is also linearized by the truncated Taylor expansion

$$J(A\mathbf{x} + \mathbf{d}) = J(\mathbf{x}) + \mathbf{g}^T(\mathbf{u}), \tag{A.13}$$

resulting in the 6×6 linear system

$$T\mathbf{z} = \mathbf{a} \tag{A.14}$$

where $\mathbf{z}^T = \begin{bmatrix} d_{xx} & d_{yx} & d_{xy} & d_{yy} & d_x & d_y \end{bmatrix}$ is the concatenation of the entries of the deformation matrix D and the displacement vector $\mathbf{d}$.

The error vector $\mathbf{a}$

$$\mathbf{a} = \int_{\mathcal{W}} [I(\mathbf{x}) - J(\mathbf{x})] \begin{bmatrix} xg_x \\ xg_y \\ yg_x \\ yg_y \\ g_x \\ g_y \end{bmatrix} w(\mathbf{x}) d\mathbf{x} \tag{A.15}$$

depends on the difference between the two images $I\,(\mathbf{x} - J(\mathbf{x}))$, and the 6×6 matrix T, which is computed from one image only and is written as

$$T = \int_{\mathcal{W}} \begin{bmatrix} U & V \\ V^T & Z \end{bmatrix} w(\mathbf{x}) d\mathbf{x} \tag{A.16}$$

where

$$U = \begin{bmatrix} x^2 g_x^2 & x^2 g_x g_y & xy g_x^2 & xy g_x g_y \\ x^2 g_x g_y & x^2 g_y^2 & xy g_x g_y & xy g_y^2 \\ xy g_x^2 & xy g_x g_y & y^2 g_x^2 & y^2 g_x g_y \\ xy g_x g_y & xy g_y^2 & y^2 g_x g_y & y^2 g_y^2 \end{bmatrix}$$

$$V^T = \begin{bmatrix} x g_x^2 & x g_x g_y & y g_x^2 & y g_x g_y \\ x g_x g_y & x g_y^2 & y g_x g_y & y g_y^2 \end{bmatrix} \tag{A.17}$$

$$Z = \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{bmatrix}.$$

Because of the Taylor series truncation (linearization), the linear system in (A.14) can only be approximately satisfied.

The correct affine model is found by iterating in a Newton-Raphson style.

While the affine model considers both the deformations and translations, Lucas and Kanade [124], warn against modeling the two problems at the same time. This is because the deformation matrix D and displacement vector **d** interact through V, and errors in the estimation of D affect **d** and vice-versa. So, attempting to determine deformation parameters is not only useless but can lead to poor displacement solutions. Frame by frame, the linear deformation of the feature window (modeled by D) is likely to be small (D should be close to the zero matrix), in contrast, the magnitude of **d** can be significant even from one frame to the next, the affine model is necessary to adapt the feature window, deforming it when perspective changes occur. It is not necessary to estimate D at every frame because window deformations are only important once enough changes have accumulated.

APPENDIX B

Matrix Information Geometry

All the concepts introduced in this chapter were taken from mathematical texts specifically designed for computer and data scientists [49],[13] and [110]. The work of Nielsen and Bhatia [100] deserving special interest.

Differential Geometric Methods

Differential Geometric Methods are being used for many different applications such as *Shape Analysis* and *Activity Recognition*. In these areas, we often need to do some comparatives between object instances and registered models, perform classification between two or more classes or to calculate the average of some samples to get a single model which somehow arrives to represent them all (to estimate a barycenter for example). For simplicity reasons, and maybe because of the academic origins of the majority of the computer vision community, the convenience of carrying out these calculations in their own spaces instead of forcing the Euclidean metrics to fit into the problems has been disregarded.

Here, we have some examples of some typical problems we often want to solve:

1. Data interpolation: Given $a, b \in M$ compute

$$(1-\lambda)a + \lambda b \qquad \lambda \in [0,1] \tag{B.1}$$

2. Compute the mean of a finite set of data $a = \{a_1 + \cdots + a_n\}$,

$$\bar{a} = \frac{a_1 + \cdots + a_n}{n} \tag{B.2}$$

3. Compute the variance of a finite set of data $a = \{a_1 + \cdots + a_n\}$,

$$var(a) = \frac{\sum_{i=1}^{n}(a_i - \bar{a})^2}{n-1} \tag{B.3}$$

4. Given two samples, $a = \{a_1, \cdots, a_n\}$ and $b = \{b_1, \cdots, b_n\}$ find their covariance,

$$cov(a,b) = \frac{\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b})}{n-1} \tag{B.4}$$

5. Assume M is some kind of geometric space. For $a, b \in M$, find a shortest path from a to b.

If M is a vector space, there are good and very popular methods for solving such problems. However, if M is a *curved space* it may be very difficult. Nonlinear manifolds are spaces that are not vector spaces, for example cases where the

addition $ax + by \notin M$ even when $x, y \in M$ and $a, b \in \mathbb{R}$. Here, the usual Euclidean calculus does not apply. The same happens for the case of conventional statistics. Analysis on nonlinear manifolds is required when certain natural constraints make the underlying space nonlinear. For example elements of $\mathbb{R}^n$ with unit norm constraint, or matrices with orthogonality constraints. Using differential geometry of the underlying manifold, one can usually derive most of the corresponding results from Euclidean spaces.

Manifolds are sets that are locally Euclidean, there exists manifolds in different types of spaces: $\mathbb{R}$, $\mathbb{R}^{n\times n}$, the general linear group $GL(n, \mathbb{R})$ (formed by $n \times n$ real invertible matrices together with their multiplication) or the set of symmetrical $n \times n$ matrices. One space of particular interest in this chapter is the space of *positive matrices* i.e.,

$$\langle x, Ax \rangle >= 0 \text{ for all } x \neq 0, \tag{B.5}$$

where $\langle x, y \rangle$ represents the inner product between two vectors x and y.

For a manifold M and a point p in M, there is a *tangent space* associated to it. This space is formed by the set of velocity vectors of all curves passing through p on M. It is known that T_pM is a vector space of the same dimension as M. Many problems in computer science and vision are solved first in T_pM and these solutions are then projected back to M using a mapping technique known as the *exponential map*: $exp_p : T_pM \longrightarrow M$ (see section B.0.5). There is an inverse to this mapping function that points from the manifold to the tangent space: $exp_p^{-1} : M \longrightarrow T_pM$.

For every point p on the manifold M, and for every vector v_1, v_2 in T_pM, there is an inner product $\langle v_1, v_2 \rangle_p$. This metric varies smoothly across p, and the usual Euclidean metrics are considered according to the tangent space type e.g., $\|\mathbf{q} - \mathbf{p}\|$ for $\mathbb{R}^n$, $tr(A^*B)$ for $\mathbb{R}^{n\times n}$.

B.0.1 A quick Tour of Basic Differential Geometry

In 1827, Carl Friedrich Gauss published the mathematical results he obtained while measuring the distance between cities and other landmarks. Gauss showed [50] that surfaces can be analyzed *extrinsically*: relating them to their embedding in the Euclidean space ($\mathbb{E}^3$ for the earth's surface surface case) or *intrinsically*: analyzing their properties which depend solely on the distance measured along curves on the surface. In the particular case of the Earth, this arises from the fact that the surface of a globe does not have the same geometry as the Euclidean plane. Today, a whole field of mathematics know as *Differential geometry* has evolved around this original idea.

Differential geometry is the theory of curved surfaces, where a surface is described by Cartesian coordinates and then analyzed employing differential calculus. This theory is the cornerstone of many interesting applications like 3D Computer graphics and Computer stereo vision.

B.0.2 Differential geometry of surfaces

Differential geometry of surfaces revolves around the fact that a curved surface can be studied without reference to a higher dimensional Euclidean space. The mathematical concept which helps to analyze intrinsically the geometry of surfaces is called the *Gaussian curvature of a surface*, or often simply the *curvature of a surface*. Care must be taken to avoid confusion with the *curvature of a curve*. The curvature of a curve is an extrinsic geometric property, telling how it is bent

in the plane, or in space. The surface of an sphere is considered to curve positively at every point in it because it bulges out in all directions. If we take any point on the surface of an sphere, there is a tangent plane touching the surface at that point only so that the rest of the surface lies all on one side except at that point. This property does not hold when considering a flat plane or the surface of a cone: for any point in them there are many other others which intersect on the tangent plane that corresponds to that point (the curvature is zero). Surfaces have negative curvature when given any point on the surface and its respective tangent plane, the surface is separated in two by the tangent plane at that point (there are points of the surface which lie on both sides of the plane).

B.0.3 Riemannian geometry

Riemannian geometry started as a very broad and abstract generalization of the differential geometry of surfaces embedded in $\mathbb{E}^3$. To each point on a curved surface corresponds a local coordinate patch on which a two-dimensional Euclidean coordinate system is defined: the tangent space in $\mathbb{E}^2$. Carrying on this idea, we arrive to the concept of the n-dimensional *Manifold* (denoted as $\mathbb{M}^n$). In mathematics, manifolds are n-dimensional topological spaces where the neighborhood of each point resembles to $\mathbb{E}^n$ (an space modeled on Euclidean spaces). A Riemannian manifold is a real smooth manifold M equipped with an inner product g_p on each tangent space T_pM that varies smoothly from point to point. A *Riemannian metric* (tensor) groups the family of g_p inner products, providing the tools required to define other geometric notions on a Riemannian manifold such as lengths or curves, angles, areas, curvature, gradients of functions and divergence of vector fields.

B.0.4 Riemannian Metrics

According to [49], if (M, g) is a Riemannian manifold, then the concept of *length* makes sense for any piecewise smooth curve on M. It is possible to define the structure of a metric space on M, where $d(p, q)$ is the greatest lower bound of the length of all curves joining p and q. Curves on M yielding the shortest distance between two points are called *geodesics*, they play an important role in many geometrical applications.

Given any point $p \in M$, for every vector v in its tangent space ($v \in T_pM$), the norm of v is defined by

$$\|v| \ = \sqrt{g_p(v, v)}. \tag{B.6}$$

The Riemannian inner product, $g_p(u, v)$, of two tangent vectors, $u, v \in T_pM$ is denoted as $\langle u, v\rangle_p$. Now, given a smooth parametric curve on M, which is a map from a closed interval $[a, b]$ in $\mathbb{R}$ so that $\gamma : [a, b] \longrightarrow M$ defines a smooth curve in M. This smooth curve can be regarded as a piecewise smooth curve from p to q iff there is a sequence $a = t_0 < t_1 < \cdots < t_{k-1} < t_k = b$ of numbers $t_i \in \mathbb{R}$, so that each map, $\gamma_i = \gamma[t_i, t_{i+1}]$ defines a *curve segment*.

The set of all piecewise smooth curves that go from p to q is denoted by $\Omega(M; p, q)$ or $\Omega(p, q)$, this set is an important object know as the *path space* of M from p to q. At any junction point, $\gamma_{i-1}(t_i) = \gamma_i(t_i)$ there may be a jump in the velocity vector of γ.

Given any curve, $\gamma \in \Omega(M; p, q)$, the *length*, $L(\gamma)$, of γ is

$$L(\gamma) = \sum_{i=0}^{k-1} \int_{t_i}^{t_i+1} \|\gamma'(t)\| dt \tag{B.7}$$

For every curve, γ, on M, let $\frac{D}{dt}$ be the associated covariant derivative along γ, also denoted $\nabla_{\gamma'}$. A curve, $\gamma : I \longrightarrow M$ is a geodesic if $\gamma'(t)$ is parallel along γ, that is, iff

$$\frac{D\gamma'}{dt} = \nabla_{\gamma'}\gamma' = 0 \tag{B.8}$$

B.0.5 The Exponential Map

An exponential map helps to parametrize a Riemannian manifold, M, locally near any $p \in M$ in terms of a map from the tangent space T_pM to the manifold.

Let (M, g) be a Riemannian manifold, for every $p \in M$, let $\mathcal{D}(p)$ be the open subset of T_pM given by

$$\mathcal{D} = \{v \in T_pM | \gamma_v(1) \text{ is defined}\}, \tag{B.9}$$

where γ_v is the unique maximal geodesic with initial conditions $\gamma_v(0) = p$ and $\gamma_v'(0) = v$. The *exponential map* is the map from $exp_p : \mathcal{D}(p) \longrightarrow M$, so that

$$exp_p(v) = \gamma_v(1). \tag{B.10}$$

This map takes a given tangent vector to the manifold, runs along the geodesic starting at that point and going in that direction, for a unit time. Since v corresponds to the velocity vector of the geodesic, the actual distance (in Riemannian terms) traveled will be dependent on that. A variation on the tangent vector v will get us different points on M resulting of the mapping exp_p.

B.0.6 Riemannian Metrics on Positive Matrices

Let $\mathbb{M}_n$ be a Hilbert space with product $\langle A, B\rangle = trA^*B$ and the associated norm $\|A\|_2 = (trA^*A)^{1/2}$. The set of Hermitian matrices constitutes a real vector space $\mathbb{H}_n$ in $\mathbb{M}_n$. The subset $\mathbb{P}_n$ consisting of strictly positive matrices is an open subset in $\mathbb{H}_n$. $\mathbb{P}_n$ is a differentiable manifold. The tangent space to $\mathbb{P}_n$ at any point $A \in \mathbb{P}_n$ is denoted as $T_A\mathbb{P}_n = \{A\} \times \mathbb{H}_n$.

The inner product in $\mathbb{H}_n$ leads to a Riemannian metric on the manifold $\mathbb{P}_n$. This metric is expressed by the differential

$$ds = \|A^{-1/2}dAA^{-1/2}\|_2 = \left[tr(A^{-1}dA)^2\right]^{1/2}. \tag{B.11}$$

Equation (B.11) is useful to compute the length of a differentiable path in $\mathbb{P}_n$. The expression $\gamma : [a, b] \to \mathbb{P}_n$ parametrically defines the path, and it is possible to get its length as

$$L(\gamma) = \int_a^b \|\gamma^{-1/2}(t)\gamma'(t)\gamma^{-1/2}\|_2 dt \tag{B.12}$$

For any two points A and B in $\mathbb{P}_n$ let

$$\delta_2(A,B) = \inf\{\mathrm{L}(\gamma){:}\gamma \text{ is a path from } A \text{ to } B\} \tag{B.13}$$

This metric in $\mathbb{P}_n$ respects the triangle inequality

$$\delta_2(A,B) \leq \delta_2(A,C) + \delta_2(C,B) \tag{B.14}$$

because the path γ_1 from A to C can be adjoined to the path γ_2 from C to B, forming the path A to B of length $L(\gamma_1) + L(\gamma_2)$.

The infimum in (B.13) is attained at a unique path joining A to B. This path is the *geodesic* from A to B.

Let $H(t), a \leq t \leq b$ be any path in $\mathbb{H}_n$ and let $\gamma(t) = e^{H(t)}$, then

$$L(\gamma) \geq \int_a^b \|H'(t)\|_2 dt. \tag{B.15}$$

If $\gamma(t)$ is any path joining A and B in $\mathbb{P}_n$, then $H(t) = \log \gamma(t)$ is a path joining $\log A$ and $\log B$ in $\mathbb{H}_n$ (the length of the path in Euclidean space). The right-hand side of equation (B.15) imposes a lower bound: the length of the straight line segment joining $\log A$ and $\log B$. This means that $L(\gamma) \geq \|\log A - \log B\|_2$.

For any two matrices H and K in $\mathbb{H}_n$ $\delta_2(e^H, e^K) \geq \|H - K\|_2$, so the map

$$(\mathbb{H}_n, \|\cdot\|_2) \xrightarrow{exp} (\mathbb{P}_n, \delta_2) \tag{B.16}$$

is metric increasing.

However, it can be proved that when A and B commute, the exponential map carries the line segment joining $\log A$ and $\log B$ in $\mathbb{H}_n$ to the geodesic joining A and B in $\mathbb{P}_n$.

Parametrically, we write $[H, K]$ for the line segment

$$H(t) = (1-t)H + tK, \qquad 0 \leq t \leq 1 \tag{B.17}$$

which joins H and K in $\mathbb{H}_n$. In the particular case where A and B are commuting matrices in $\mathbb{P}_n$ the distance

$$\delta_2(A,B) = \|\log A - \log B\|_2. \tag{B.18}$$

represents the geodesic from A to B.

It is possible to get the length of any subinterval $[0, a]$ of $[0, 1]$ thanks to the parametrization

$$H(t) = (1-t)\log A + t \log B. \tag{B.19}$$

Similarly, the natural parametrization of the geodesic $[A, B]$ when A and B commute is

$$\gamma(t) = A^{1-t}B^t, \quad 0 \leq t \leq, \tag{B.20}$$

which means that $\delta_2(A, \gamma(t)) = t\delta_2(A, B)$ for each t.

In the general case (when A and B can be any two elements of $\mathbb{P}_n$), there exists a unique *geodesic* $[A, B]$ joining A and B. This geodesic has a parametrization

$$\gamma(t) = A^{1/2}\left(A^{-1/2}BA^{-1/2}\right)^t A^{1/2}, \quad 0 \leq t \leq 1, \tag{B.21}$$

expressed as

$$\delta_2(A, \gamma(t)) = t\delta_2(A, B) \tag{B.22}$$

for each t.

From this parametrization, it can be proved that the geodesic length is

$$\delta_2(A, B) = \| \log A^{-1/2}BA^{-1/2}\|_2 \tag{B.23}$$

Equation (2.27) is the *Riemannian metric* on the manifold $\mathbb{P}_n$. Applying the definition of the norm $\| \cdot \|_2$ we have

$$\delta_2(A, B) = \left(\sum_{i=1}^{n} \log^2 \lambda_i \left(A^{-1}B\right)\right)^{1/2} \tag{B.24}$$

where λ_i are the eigenvalues of the matrix $A^{-1}B$.

B.0.7 Model Update Techniques

B.0.7.1 Symmetric Positive Definite Matrices (*SPD*) Averages

Because of occlusions and appearance changes, tracking objects for the long-term is a difficult problem. To identify occlusions and follow appearance changes, a robust algorithm must adapt its object description. Such updating mechanism needs to recognize the relevant information available from all the description samples. It must be insensible to abrupt changes, and should consider all the information at hand to improve object discrimination.

Covariance matrices are a subset of the symmetric positive definite matrices (*SPD*) set. The averaging of this type of matrices is an important and open problem which arises when one has to represent as closely as possible(and through a single matrix) the information contained in several $n \times n$ positive matrices $A_1, \cdots A_m$. *Matrix Information Geometry* is the field of mathematics dedicated to these type of problems. Good sources of information from this area of mathematics are [100] and [13].

The update the model, the straightforward choice would be the use of the arithmetic mean of $A_1, \cdots, A_m$

$$\frac{1}{m}\sum_{j=1}^{m} A_j \tag{B.25}$$

but this mean does not satisfy some expected properties and gives poor results. As we will see, all the desired properties are satisfied by the geometric mean, justifying the need of defining a suitable geometric mean of *SPD* matrices.

A mechanism to update the covariance model was presented in 2006 by Porikli et al. in [109], but there are other options at our disposal. Most of these options are based on the estimation of the *geometric mean*. In this section a novel method based on the *geometric median* is introduced to protect the algorithm against outliers presence.

B.0.7.2 Binary Geometric Mean

Let $\mathbb{R}_+$ be the set of positive numbers. A *mean* is a function $m : \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ that satisfies the following conditions

$$\begin{aligned}
&\text{(i) } m(a,b) = m(b,a)\\
&\text{(ii) } \min(a,b) \leq m(a,b) \leq \max(a,b)\\
&\text{(iii) } m(\alpha a, \alpha b) \leq \alpha m(a,b) \text{ for all } \alpha > 0.\\
&\text{(iv) } a \leq a' \Rightarrow m(a,b) \leq m(a',b)\\
&\text{(v) } m \text{ is continuous.}
\end{aligned} \tag{B.26}$$

The arithmetic, geometric and harmonic means defined as

$$\frac{a+b}{2}, \sqrt{ab}, \left(\frac{a^{-1}+b^{-1}}{2}\right)^{-1} \tag{B.27}$$

respectively, are the most familiar examples of means satisfying the conditions enlisted in (B.26). But there are several others such as the *logarithmic mean*

$$L(a,b) = \frac{a-b}{\log a - \log b} = \int_0^1 a^{1-t} b^t dt, \tag{B.28}$$

and the binomial means

$$B_p(a,b) = \left(\frac{a^p + b^p}{2}\right)^{1/p}, -\infty < p < \infty. \tag{B.29}$$

Some particular cases of (B.29) are

$$\lim_{p\to 0} B_p(a,b) = \sqrt{ab}, \tag{B.30}$$

$$\lim_{p\to \infty} B_p(a,b) = \max(a,b), \tag{B.31}$$

and

$$\lim_{p\to -\infty} B_p(a,b) = \min(a,b). \tag{B.32}$$

As stated before, the set of $n \times n$ covariance matrices is a subset of the positive matrices $\mathbb{P}(n)$ set. The five conditions

of (B.26) can be imitated to the define the concept of *matrix mean* which is a map $M : \mathbb{P}(n) \times \mathbb{P}(n) \to \mathbb{P}(n)$ satisfying

$$\begin{aligned}
&\text{(i)' } M(A,B) = M(B,A).\\
&\text{(ii)' If } A \leq B, \text{then } A \leq M(A,B) \leq B.\\
&\text{(iii)' } M(X^*AX, X^*BX) = X^*M(A,B)X, \text{ for all nonsingular matrices } X\\
&\text{(iv)' } A \leq A' \Rightarrow M(A,B) \leq M(A',B)\\
&\text{(v)' } M \text{ is continuous.}
\end{aligned} \tag{B.33}$$

Several problems arise for the list of conditions in (B.33). For example, matrix multiplication is non commutative, the order relation $A \leq B$ defined for the case when the difference $B - A$ is positive definite does not necessarily imply $A^2 \leq B^2$.

The arithmetic and harmonic means

$$\frac{A+B}{2}, \left(\frac{A^{-1}+B^{-1}}{2}\right)^{-1} \tag{B.34}$$

do meet the five conditions in (B.33), but the geometric mean have some peculiarities. The matrix $A^{1/2}B^{1/2}$ is not positive, not even Hermitian unless A and B commute.

The analogy of (B.30) for the case of positive matrices is

$$\lim_{p\to 0}\left(\frac{A^p+B^p}{2}\right)^{1/p}, \tag{B.35}$$

or

$$\exp\left(\frac{\log A + \log B}{2}\right). \tag{B.36}$$

While these matrices are positive, they do not obey properties (iii)' and (iv)' because the exponential map is not order-preserving, and $A \mapsto A^t$ is order preserving if and only if $0 \leq t \leq 1$.

The operation

$$A\#B := A^{1/2}\left(A^{-1/2}BA^{-1/2}\right)^{1/2}A^{1/2} \tag{B.37}$$

satisfies all the desired properties (i)'-(v)'. These operation was presented in 1975 by Pusz and Woronowicz [112] and is valid for the binary case (two matrices problem) only. The extension for more than two positive matrices turned out to be a tricky problem which resisted solution for more than 25 years. The solution to this problem has been found recently, the approach (introduced in the next subsection) involves some differential geometry.

B.0.7.3 Riemannian Mean

The *Riemannian barycenter center of mass* of m elements $A_1, A_2, \cdots, A_m$ is defined as

$$G(A_1, A_2, \cdots, A_m) = \arg\min \sum_{j=1}^{m} \delta_2^2(X, A_j) \tag{B.38}$$

where $\arg\min f(X)$ returns the minimum point X_0 in the function $f(X)$. It can be shown that the solution of the matrix equation

$$\sum_{j=1}^{m} \log\left(A_j^{-1/2} X A_j^{-1/2}\right) = 0 \tag{B.39}$$

provides us the value of this point.

In a Euclidean space the averages s_j of a vector $a_1, \cdots, a_m$ are defined as

$$\begin{aligned} s_1 &= a_1 \\ s_2 &= \frac{1}{2}(a_1 + a_2) \\ s_3 &= \frac{2}{3}s_2 + \frac{1}{3}a_3 = \frac{1}{3}(a_1 + a_2 + a_3) \\ s_k &= \frac{k-1}{k}s_{k-1} + \frac{1}{k}a_k. \end{aligned} \tag{B.40}$$

A procedure inspired in the Euclidean case can be ported to calculate positive matrices averages

$$\begin{aligned} S_1 &= A_1 \\ S_2 &= (A_1 \#_{1/2} A_2) \\ S_3 &= S_2 \#_{1/3} A_3 \\ S_k &= S_{k-1} \#_{1/k} A_k \end{aligned} \tag{B.41}$$

It is not possible to expect that S_m would be the Riemannian mean $G(A_1, \cdots, A_m)$. However, there is an adaptation of this idea which provides a sequence that converges to G.

Given a sequence of independent trials in which an integer is chosen from the set $\{1, 2, \cdots, m\}$ with equal probability. Thus, a sequence of integers represented by $\mathcal{I} = \{i_1, i_2, \cdots\}$. Now, let $\{S_k(\mathcal{I}, A)\}$ be the sequence of binary means

$$S_1 = A_{i_1}, S_2 = S_1 \#_{1/2} A_{i_2}, \cdots, S_k = S_{k-1} \#_{1/k} A_{i_k}. \tag{B.42}$$

In [129] it is shown that for almost all $\mathcal{I}$ its corresponding sequence $\{S_k(\mathcal{I}, A)\}$ converges to $G(A_1, \cdots, A_m)$. The Riemannian mean $G(A_1, \cdots, A_m)$ is the limit of a sequence constructed from $A_1, \cdots, A_m$ by taking at each step a binary geometric mean.

One problem with this approach is that it is not deterministic, Holbrook solved this issue in [65], he has shown that it is possible to reach the barycenter $G(A_1, \cdots, A_m)$ as a limit of a *deterministic walk*, for any $X \in (P)(n)$ let $\varphi(X) = X \#_{1/r} A_k$ where $k = r(\mod m)$. Let the sequence

$$\varphi_{r,n} = \varphi_{r+n-1} \cdots \varphi_{r+1} \cdot \varphi_r. \tag{B.43}$$

For all X, and for all positive integers r,

$$\lim_{n\to\infty} \varphi_{r,n}(X) = G(A_1, \cdots, A_m) \tag{B.44}$$

As an example, choosing $X = A_1$ and $r = 1$, the sequence in (B.44) develops as

$$((((A_1 \#_{1/2} A_2) \#_{1/3} A_3) \cdots \#_{1/m} A_m) \#_{1/m+1} A_1) \#_{1/m+2} A_2 \cdots$$

which converges to $G(A_1, \cdots, A_m)$. This is true, because the distance of $\varphi_{r,n}(X)$ from G is reduced after every m steps.

B.0.7.4 Riemannian Geometric Median

If M is Riemannian manifold and given some samples $x_1, x_2, \cdots, x_N \in M$ and their associated weights $w_1, w_2, \cdots, w_N$ where $\sum_i w_i = 1$. The weighted sum of distances is

$$f(x) = \sum_i w_i d(x, x_i) \tag{B.45}$$

where d is the Riemannian distance function in M. Assuming that x_i lies in a convex set $U \subset M$, then any pair of points in U is connected by a unique shortest geodesic contained entirely in U.

The *weighted geometrical median* is the minimizer of $f(x)$ (equation B.45)

$$m = \arg\min_{x \in M} \sum_i^N w_i d(x, x_i), \tag{B.46}$$

when all the weights are equal ($w_i = \frac{1}{N}$), equation (B.46) represents the *geometric median*.

It is important to remark the difference between the *weighted geometrical median* defined in equation (B.46) from the Fréchet/Karcher mean which is a generalization of the Euclidean minimum squares principle into Riemannian geometry as

$$\mu = \arg\min_{x \in M} \sum_i^N w_i d^2(x, x_i) \tag{B.47}$$

For a Riemannian manifold M, the gradient of the Riemannian sum-of-distances function is given by

$$\nabla f(x) = -\sum_{i=1}^N w_i Log_x(x_i)/d(x, x_i), \tag{B.48}$$

this gradient is not defined for x is any of the data points x_i.

An iterative steepest descent method is used to find the Riemannian geometric median when it exists

$$\begin{aligned} m_{k+1} &= Exp_{m_k}(\alpha v_k), \\ v_k &= \sum_{i \in I_k} \frac{w_i Log_{m_k}(x_i)}{d(m_k, x_i)} \cdot \left(\sum_{i \in I_k} \frac{w_i}{d(m_k, x_i)} \right)^{-1}. \end{aligned} \tag{B.49}$$

B.1 Covariance descriptor computation

From 1.38, the (i, j)-th element of the covariance matrix is

$$C_R(i,j) = \frac{1}{n-1}\sum_{k=1}^{n}(z_k(i) - \mu(i))(z_k(j) - \mu(j)). \tag{B.50}$$

Expanding the means and rearranging the terms we have

$$C_R(i,j) = \frac{1}{n-1}\left[\sum_{k=1}^{n} z_k(i)z_k(j) - \frac{1}{n}\sum_{k=1}^{n} z_k(i)\sum_{k=1}^{n} z_k(j)\right]. \tag{B.51}$$

The covariance in a given region depends on the sum of each feature dimension $z(i)_{i=1\cdots n}$, as well as the sum of the multiplication of any pair of features $z(i)z(j)_{i,j=1\cdots n}$, requiring in total $d + d^2$ integral images, one for each feature dimension $z(i)$ and one for the multiplication of any pair of feature dimensions $z(i)z(j)$.

Let P be a $W \times H \times d$ tensor of the integral images of each feature dimension

$$P(x', y', i) = \sum_{x<x', y<y'} F(x, y, i) \text{ for } i = i \cdots d \tag{B.52}$$

and Q be the $W \times H \times d \times d$ tensor containing the feature product-pair integral images

$$Q(x', y', i, j) = \sum_{x<x', y<y'} F(x, y, i)F(x, y, j) \text{ for } i, j = i \cdots d. \tag{B.53}$$

Now, let $\mathbf{p}_{x,y}$ be a d dimensional vector and $\mathbf{Q}_{x,y}$ a $d \times d$ dimensional matrix such as

$$\begin{aligned} \mathbf{p}_{x,y} &= [P(x, y, 1) \cdots P(x, y, d)]^T \\ \mathbf{Q}_{x,y} &= \begin{pmatrix} Q(x, y, 1, 1) & \cdots & Q(x, y, 1, d) \\ & \vdots & \\ Q(x, y, d, 1) & \cdots & Q(x, y, d, d) \end{pmatrix} \end{aligned} \tag{B.54}$$

Let a region $R((x', y'); (x'', y''))$ be the rectangular region defined by the top-left point (x', y') and the right-bottom point (x'', y''). The covariance of the region bounded by (1,1) and (x', y') is

$$\mathbf{C}_R((1, 1); (x', y')) = \frac{1}{n-1}\left[\mathbf{Q}_{x',y'} - \frac{1}{n}\mathbf{p}_{x',y'}\mathbf{p}^T_{x',y'}\right] \tag{B.55}$$

where $n = x' \cdot y'$. Similarly and after some algebraic manipulations, the covariance of the region $R((x', y'); (x'', y''))$

is

$$\mathbf{C}_{R((x',y');(x'',y''))} = \frac{1}{n-1}\Big[\mathbf{Q}_{x'',y''} + \mathbf{Q}_{x',y'} - \mathbf{Q}_{x'',y'} - \mathbf{Q}_{x',y''} - \frac{1}{n}\left(\mathbf{p}_{x'',y''} + \mathbf{p}_{x',y'} - \mathbf{p}_{x',y''} - \mathbf{p}_{x'',y'}\right)\left(\mathbf{p}_{x'',y''} + \mathbf{p}_{x',y'} - \mathbf{p}_{x',y''} - \mathbf{p}_{x'',y'}\right)^T\Big] \tag{B.56}$$

where $n = (x'' - x') \cdot (y'' - y')$. After constructing the integral images the covariance of any rectangular region can be computed in $O(d^2)$ time regardless of the size of the region $R((x', y'); (x'', y''))$. The complete process is represented graphically in **Figure B-1**.

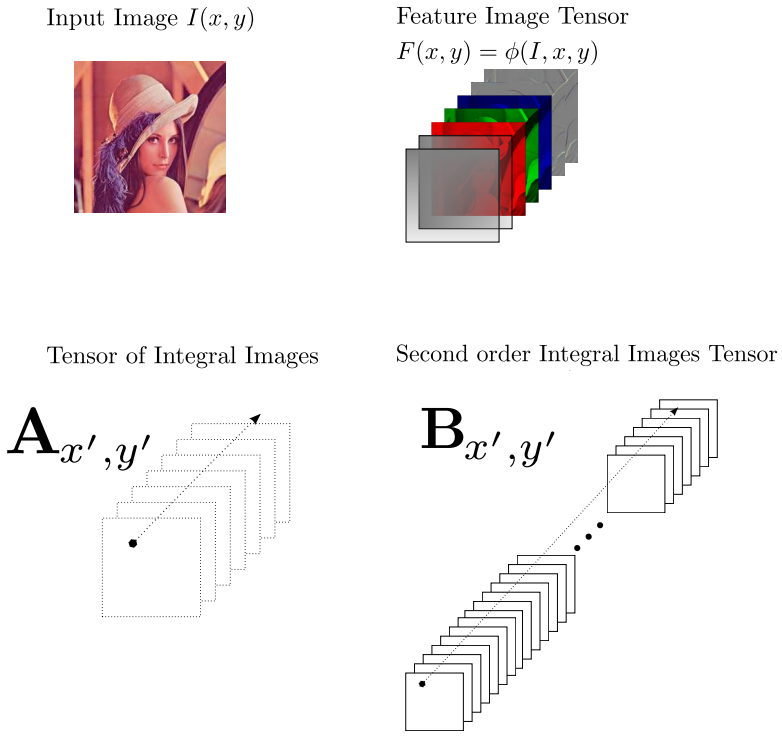


Figure B-1: Covariance Description Calculation

B.2 Gradient descent in covariance matrices

Let us first define the term $P(\mathbf{x}) = \left(\mathbf{M}^{-\frac{1}{2}}\mathbf{Y}_\mathbf{x}\mathbf{M}^{-\frac{1}{2}}\right)$, the gradient of $f(\mathbf{x})$ in equation (2.35) is defined as

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \partial_x f(\mathbf{x}) & \partial_y f(\mathbf{x}) \end{bmatrix}^T \tag{B.57}$$

where,

$$\begin{aligned} \partial_x f(\mathbf{x}) &= \partial_x d^2(\mathbf{M}, \mathbf{Y}_\mathbf{x}) \\ &= \partial_x \text{tr}\left[\log^2 P(\mathbf{x})\right] \\ &= \text{tr}\left[\partial_x log^2 P(\mathbf{x})\right] \\ &= \text{tr}\left[2\log P(\mathbf{x}) P(\mathbf{x})^{-1} \partial_x P(\mathbf{x})\right] \end{aligned} \tag{B.58}$$

so that,

$$\partial_y f(\mathbf{x}) = \text{tr}\left[2\log P(\mathbf{x})P(\mathbf{x})^{-1}\partial_y P(\mathbf{x})\right] \tag{B.59}$$

where,

$$\begin{aligned}\partial_x P(\mathbf{x}) &= \partial_x(\mathbf{M}^{-\frac{1}{2}}\mathbf{Y}_\mathbf{x}\mathbf{M}^{-\frac{1}{2}}) \\ &= \left(\mathbf{M}^{-\frac{1}{2}} \otimes \mathbf{M}^{-\frac{1}{2}}\right)\text{vec}(\partial_x \mathbf{Y}_\mathbf{x}) \\ &= \mathbf{M}^{-\frac{1}{2}}(\partial_x \mathbf{Y}_\mathbf{x})\mathbf{M}^{-\frac{1}{2}}\end{aligned} \tag{B.60}$$

where the operator $\otimes$ denotes the Kronecker matrix product.

Partial derivatives $(\partial_x \mathbf{Y}_\mathbf{x})$ and $(\partial_y \mathbf{Y}_\mathbf{x})$ can be approximated from discrete data as

$$\begin{aligned}\partial_x \mathbf{Y}_\mathbf{x} &\approx (\mathbf{Y}_{\mathbf{x}|x+dx,y} - \mathbf{Y}_{\mathbf{x}|x-dx,y})/2dx \\ \partial_y \mathbf{Y}_\mathbf{x} &\approx (\mathbf{Y}_{\mathbf{x}|x,y+dy} - \mathbf{Y}_{\mathbf{x}|x,y-dy})/2dy\end{aligned} \tag{B.61}$$

where $dx = dy = 1$.

The subtraction operator in the partial equations (B.61) needs to be replaced by its equivalent in the SPD space

$$\begin{aligned}\partial_x \mathbf{Y}_\mathbf{x} &\approx 0.5\log_{\mathbf{Y}_\mathbf{x}|x-dx,y}(\mathbf{Y}_{\mathbf{x}|x+dx,y})/dx \\ \partial_y \mathbf{Y}_\mathbf{x} &\approx 0.5\log_{\mathbf{Y}_\mathbf{x}|x,y-dy}(\mathbf{Y}_{\mathbf{x}|x,y+dy})/dy\end{aligned} \tag{B.62}$$

Once all the terms required by equation (B.57) to obtain the gradient have been given, the target location at iteration $i+1$ is obtained as

$$\mathbf{x}^{i+1} = \mathbf{x}^i - \eta^i \nabla f(\mathbf{x}^i) \tag{B.63}$$

which is the SPD space steepest descent algorithm. The value of η^i controls the step of the descent (learning rate), it can follow an annealing schedule expressed as $\eta^i = \eta^0(1 - i/N)$, where N is determined empirically. Iterations will continue until convergence, which occurs when $\|\eta^i \nabla f(\mathbf{x}^i)\| < t_{conv}$.

B.3 Pedestrian re-identification tables

Table B.1: Re-identification pattern of covariances descriptors.

# region	**Top-left** (x, y)	**Bottom-right** (x, y)	# region	**Top-left** (x, y)	**Bottom-right** (x, y)
1	(95,39)	(47, 87)	23	(59,38)	(35, 63)
2	(83,60)	(35, 108)	24	(70,44)	(46, 69)
3	(59,60)	(11, 108)	25	(67,53)	(47, 73)
4	(47,39)	(1, 87)	26	(62,62)	(42, 82)
5	(59,18)	(11, 66)	27	(52,62)	(32, 82)
6	(83,18)	(35, 66)	28	(47,53)	(27, 73)
7	(83,54)	(45, 92)	29	(52,44)	(32, 64)
8	(66,63)	(28, 102)	30	(62,44)	(42, 64)
9	(49,54)	(11, 92)	31	(63,59)	(46, 76)
10	(49,34)	(11, 72)	32	(55,63)	(39, 80)
11	(66,24)	(28, 63)	33	(48,59)	(31, 76)
12	(83,34)	(45, 72)	34	(48,50)	(31, 67)
13	(78,48)	(47, 78)	35	(55,46)	(39, 63)
14	(70,61)	(40, 92)	36	(63,50)	(46, 67)
15	(54,61)	(24, 92)	37	(63,55)	(47, 71)
16	(47,48)	(16, 78)	38	(59,62)	(43, 78)
17	(54,34)	(24, 65)	39	(51,62)	(35, 78)
18	(70,34)	(40, 65)	40	(47,55)	(31, 71)
19	(70,57)	(46, 82)	41	(51,48)	(35, 64)
20	(59,63)	(35, 88)	42	(59,48)	(43, 64)
21	(48,57)	(24, 82)	43	(47,51)	(23, 75)
22	(48,44)	(24, 69)			

Table B.2: Re-identification pattern of covariances for ELBCM-based descriptors.

# region	**Top-left** (x, y)	**Bottom-right** (x, y)
1	(5,5)	(91,124)
2	(5,5)	(48,124)
3	(48,5)	(91,124)
4	(5,5)	(91,65)
5	(5,65)	(91,124)

APPENDIX C

Computer architecture

C.1 Loop transformations

- **Loop interchange**: This technique is particularly important when a program is processing multidimensional arrays (such as pixel data in an image). Because of the way 2-dimensional arrays are mapped to 1-dimensional memory arrays in hardware it is very important to pay attention to which dimension is assigned to the inner loop and which dimension is assigned to the outer loop. An example is given in **Listing 7**, if the array is traversed along the wrong axis, only one element of each cache line will be used before before the program continues to the next cache line.

```
/* Before */
  for (k = 0; k < 100; k = k + 1)
    for (j = 0; j < 100; j = j + 1)
      for (i = 0; i < 5000; i = i + 1)
        x[i][j] = 2 * x[i][j];
```

```
/* After */
  for (k = 0; k < 100; k = k + 1)
    for (i = 0; i < 5000; i = i + 1)
      for (j = 0; j < 100; j = j + 1)
        x[i][j] = 2 * x[i][j];
```

Listing 7: Loop interchange example.

- **Loop fusion**: Many programs have separate loops that operate on the same data, combining these loops it is possible to take advantage of the temporal locality by grouping operations on the same data together. An example of this is provided in **Listing 8**, in the original version, the first loop accesses the whole data set before the next loop can start producing multiple sweeps through the same cache lines. If the working set does not fit into the cache performance is reduced dramatically. The second version observe that it is not necessary to wait for all the operations cycles of the first loop to complete before starting the operations of the second loop, loops are merged to reduce the number of passes through the working set.

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    a[i][j] = 1/b[i][j] * c[i][j];

for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    d[i][j] = a[i][j] + c[i][j];
```

```
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {
    a[i][j] = 1/b[i][j] * c[i][j];
    d[i][j] = a[i][j] + c[i][j];
  }
```

Listing 8: Loop fusion example. The code on the left is optimized to increase temporal locality and reduce the number of cache misses when accessing arrays `a` and `c` from two misses per access to just one.

Bibliography

[1] Alexandre Alahi, Raphaël Ortiz, and Pierre Vandergheynst. FREAK: Fast Retina Keypoint. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[2] M. Andriluka, S. Roth, and B. Schiele. People-tracking-by-detection and people-detection-by-tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1--8. IEEE, 2008.

[3] A. Andriyenko and K. Schindler. Multi-target tracking by continuous energy minimization. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1265--1272. IEEE, 2011.

[4] A. Andriyenko, K. Schindler, and S. Roth. Discrete-continuous optimization for multi-target tracking. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1926--1933. IEEE, 2012.

[5] R. V. Babu, P. Pérez, and P. Bouthemy. Robust tracking with motion estimation and local kernel-based color modeling. *IVC*, 25(8), 2007.

[6] S. Bak, E. Corvee, F. Bremond, and M. Thonnat. Multiple-shot human re-identification by mean riemannian covariance grid. In *Advanced Video and Signal-Based Surveillance (AVSS), 2011 8th IEEE International Conference on*, pages 179--184. IEEE, 2011.

[7] S. Baker, D. Scharstein, JP Lewis, S. Roth, M.J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1--8. IEEE, 2007.

[8] Simon Baker, Raju Patil, German Cheung, and Iain Matthews. Lucas-kanade 20 years on: Part 1. Technical report, 2004.

[9] A. Barachant, S. Bonnet, M. Congedo, C. Jutten, *et al.* Bci signal classification using a riemannian-based kernel. In *Proceeding of the 20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 97--102, 2012.

[10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision--ECCV 2006*, pages 404--417. Springer, 2006.

[11] Peter Belhumeur, João Hespanha, and David Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Computer Vision—ECCV'96*, pages 43--58, 1996.

[12] Jerome Berclaz, Francois Fleuret, Engin Turetken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(9):1806--1819, 2011.

[13] R. Bhatia. *Positive Definite Matrices*. Princeton Series in Applied Mathematics. Princeton University Press, 2008. ISBN 9780691129181. URL `http://books.google.fr/books?id=-KIFglY18nYC`.

[14] Martin Bichsel. *Strategies of robust object recognition for the automatic identification of human faces*. PhD thesis, Diss. Naturwiss. ETH Zürich, Nr. 9467, 1991. Ref.: O. Kübler; Korref.: P. Seitz, 1991.

[15] D.A. Bini and B. Iannazzo. Computing the karcher mean of symmetric positive definite matrices. *Linear Algebra and its Applications*, 2011.

[16] S. T. Birchfield and S. Rangarajan. Spatiograms versus histograms for region-based tracking. In *CVPR*, pages II: 1158--1163, 2005. URL `http://dx.doi.org/10.1109/CVPR.2005.330`.

[17] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. ISBN 9780387310732. URL `http://books.google.fr/books?id=kTNoQgAACAAJ`.

[18] A. Blake, P. Kohli, and C. Rother. *Markov Random Fields for Vision and Image Processing*. Mit Press, 2011. ISBN 9780262015776. URL `http://books.google.fr/books?id=dQE5k7DY-dAC`.

[19] J.Y. Bouguet *et al.* Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. *Intel Corporation, Microprocessor Research Labs, OpenCV Documents*, 3, 1999.

[20] Michael D Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Online multi-person tracking-by-detection from a single, uncalibrated camera. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(9):1820--1833, 2011.

[21] With contributions by Gertjan J. Burghouts, Theo Gevers, Arjan Gijsenij, Joost van de Weijer, and Jan-Mark Geusebroek. *Color in Computer Vision*. John Wiley & Sons, Inc., 2012. ISBN 9781118350089. URL `http://dx.doi.org/10.1002/9781118350089.ch14`.

[22] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679--698, 1986.

[23] T. Carron. *Segmentation d'images couleur dans la base Teinte Luminance Saturation : approche numérique et symbolique*. Thesis, Université de la Savoie, France, December 1995.

[24] Thierry Carron and Patrick Lambert. Color edge detector using jointly hue, saturation and intensity. In *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, volume 3, pages 977--981. IEEE, 1994.

[25] B. Chapman, G. Jost, and R. van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming*. Number v. 10 in Scientific Computation Series. MIT Press, 2008. ISBN 9780262533027. URL `http://books.google.fr/books?id=MeFLQSKmaJYC`.

[26] Srinivas Chellappa, Franz Franchetti, and Markus Püschel. How to write fast numerical code: A small introduction. In *Generative and Transformational Techniques in Software Engineering II*, pages 196--259. Springer, 2008.

[27] L Chen. The lambda-connected segmentation and the optimal algorithm for split-and-merge segmentation. *Chin J Comput*, 14:321--331, 1991.

[28] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790--799, 1995.

[29] A. Cherian, S. Sra, A. Banerjee, and N. Papanikolopoulos. Efficient similarity search for covariance matrices via the jensen-bregman logdet divergence. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2399--2406. IEEE, 2011.

[30] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell*, 24(5):603--619, 2002. URL `http://doi.ieeecomputersociety.org/10.1109/34.1000236`.

[31] T.H. Cormen. *Algorithms Unlocked*. Computer Science. MIT Press, 2013. ISBN 9780262518802. URL `http://books.google.fr/books?id=r-oGPLclJc0C`.

[32] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273--297, 1995.

[33] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *Electronic Computers, IEEE Transactions on*, (3):326--334, 1965.

[34] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886--893. IEEE, 2005.

[35] Silvano Di Zenzo. A note on the gradient of a multi-image. *Computer Vision, Graphics, and Image Processing*, 33 (1):116--125, 1986.

[36] Paul Doe. Performance analysis guide for intel core i7 processor and intel xeon 5500 processors. URL `http://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf`.

[37] M. Domeika. *Software Development for Embedded Multi-core Systems: A Practical Guide Using Embedded Intel Architecture*. Elsevier Science, 2011. ISBN 9780080558585. URL `http://books.google.fr/books?id=Lyw3K_RJiWMC`.

[38] Gabriele Facciolo, Nicolas Limare, and Enric Meinhardt. Preprint january 23, 2013 integral images for block matching.

[39] J. Ferryman and A. Shahrokni. Pets2009: Dataset and challenge. In *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, pages 1 --6, dec. 2009. doi: 10.1109/PETS-WINTER.2009.5399556.

[40] P.T. Fletcher, S. Venkatasubramanian, and S. Joshi. Robust statistics on riemannian manifolds via the geometric median. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1--8. IEEE, 2008.

[41] Michael J Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, 100(9): 948--960, 1972.

[42] Itzhak Fogel and Dov Sagi. Gabor filters as texture discriminator. *Biological cybernetics*, 61(2):103--113, 1989.

[43] W. Förstner. A feature based correspondence algorithm for image matching. *International Archives of Photogrammetry and Remote Sensing*, 26(3):150--166, 1986.

[44] Wolfgang Förstner and Boudewijn Moonen. A metric for covariance matrices. *Quo vadis geodesia*, pages 113--128, 1999.

[45] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Pearson, 2011. ISBN 9780136085928. URL `http://books.google.fr/books?id=gM63QQAACAAJ`.

[46] Yoav Freund and Robert E Schapire. Experiments with a new boosting algorithm. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 148--156. MORGAN KAUFMANN PUBLISHERS, INC., 1996.

[47] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119--139, 1997.

[48] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[49] Jean Gallier. Lectures on geometric methods for computer science. University of Pennsylvania Lectures CIS610, 2011.

[50] C.F. Gauss, A. Hiltebeitel, and J. Morehead. *General Investigations of Curved Surfaces*. Raven Series in Higher Mathematics. Raven Press, 1965. URL `http://books.google.fr/books?id=tEDnuQAACAAJ`.

[51] Jan-Mark Geusebroek, Rein Van Den Boomgaard, Arnold WM Smeulders, and Anuj Dev. Color and scale: The spatial structure of color images. In *Computer Vision-ECCV 2000*, pages 331--341. Springer, 2000.

[52] J.M. Geusebroek, R. van den Boomgaard, A.W.M. Smeulders, and H. Geerts. Color invariance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1338--1350, 2001. ISSN 0162-8828.

[53] J.M. Geusebroek, G.J. Burghouts, and A.W.M. Smeulders. The Amsterdam library of object images. *International Journal of Computer Vision*, 61(1):103--112, 2005. ISSN 0920-5691.

[54] T. Gevers and A. W. M. Smeulders. Color-based object recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 32(3):453--464, 1999. URL `http://www.science.uva.nl/research/publications/1999/GeversIJPRAI1999`.

[55] M. Gouiffes. Tracking by combining photometric normalization and color invariants according to their relevance. In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 6, pages VI--145--VI--148, 16 2007-oct. 19 2007. doi: 10.1109/ICIP.2007.4379542.

[56] Michèle Gouiffès, Florence Laguzet, and Lionel Lacassagne. Projection-histograms for mean-shift tracking. In *ICIP*, pages 4617--4620. IEEE, 2010. ISBN 978-1-4244-7994-8. URL `http://dx.doi.org/10.1109/ICIP.2010.5651317`.

[57] K. Grauman and B. Leibe. *Visual Object Recognition*. Synthesis Lectures on Artificial Intelligence and Machine Learning Series. Morgan & Claypool, 2011. ISBN 9781598299687. URL http://books.google.fr/books?id=lAQGBvdm3UsC.

[58] Douglas Gray, S. Brennan, and H. Tao. Evaluating appearance models for recognition, reacquisition, and tracking. In *10th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS)*, 09/2007 2007.

[59] Song Guo and Qiuqi Ruan. Facial expression recognition using local binary covariance matrices. In *Wireless, Mobile & Multimedia Networks (ICWMMN 2011), 4th IET International Conference on*, page 237–242, 2011.

[60] M.J. Hannah. Computer matching of areas in stereo images. Technical report, DTIC Document, 1974.

[61] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[62] Eric Hayman, Barbara Caputo, Mario Fritz, and Jan-Olof Eklundh. On the significance of real-world conditions for material classification. *Computer Vision-ECCV 2004*, pages 253--266, 2004.

[63] N.J. Higham. *Accuracy and Stability of Numerical Algorithms*. EngineeringPro collection. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2002. ISBN 9780898718027. URL http://books.google.fr/books?id=7J52J4GrsJkC.

[64] Martin Hofmann, Michael Haag, and Gerhard Rigoll. Unified hierarchical multi-object tracking using global data association. PETS, 2013.

[65] John Holbrook. No dice: a determine approach to th cartan centroid. *Ramanujan Mathematical Society*, 2012.

[66] B.K.P. Horn and B.G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1):185--203, 1981.

[67] Fran□ois Irigoin, Pierre Jouvelot, and R□mi Triolet. Semantical interprocedural parallelization: an overview of the pips project. In *ICS*, pages 244--251, 1991. doi: http://doi.acm.org/10.1145/109025.109086.

[68] Michael Isard and Andrew Blake. Condensation—conditional density propagation for visual tracking. *International journal of computer vision*, 29(1):5--28, 1998.

[69] Sadeep Jayasumana, Richard Hartley, Mathieu Salzmann, Hongdong Li, and Mehrtash Harandi. Kernel methods on the riemannian manifold of symmetric positive definite matrices.

[70] J. Jeyakar, R. V. Babu, and K. R. Ramakrishnan. Robust object tracking using local kernels and background information. In *ICIP*, pages V: 49--52, 2007. URL http://dx.doi.org/10.1109/ICIP.2007.4379762.

[71] M. Jones and P. Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR-20003-96*, 3, 2003.

[72] K. Jungling and M. Arens. Detection and tracking of objects with direct integration of perception and expectation. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1129--1136. IEEE, 2009.

[73] Z Kalal, J Matas, and K Mikolajczyk. P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints. *Conference on Computer Vision and Pattern Recognition*, 2010.

[74] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *2010 International Conference on Pattern Recognition*, pages 2756--2759. IEEE, 2010.

[75] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(7):1409--1422, 2012.

[76] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321--331, 1988.

[77] L Lacassagne, A Manzanera, and A Dupret. Motion detection: Fast and robust algorithms for embedded systems. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3265--3268. IEEE, 2009.

[78] F. Laguzet, M. Gouiffès, and L. Lacassagne. Automatic colorspace switching for robust tracking. In *IEEE ICSIPA*, November 2011.

[79] F. Laguzet, A. Romero Mier y Terán, M. Gouiffès, and L. Lacassagne. Color tracking with contextual switching: Real-time implementation on CPU. *Journal of Real Time Image Processing (JRTIP)*, (Special Issue on Real-Time Color Image Processing), 2013.

[80] Florence Laguzet, Michèle Gouiffès, Lionel Lacassagne, and Daniel Etiemble. Automatic color space switching for robust tracking. In *ICSIPA*, pages 295--300. IEEE, 2011. ISBN 978-1-4577-0243-3. URL `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6138563`.

[81] Bastian Leibe, Konrad Schindler, and Luc Van Gool. Coupled detection and trajectory estimation for multi-object tracking. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1--8. IEEE, 2007.

[82] Peihua Li and Qilong Wang. Local log-euclidean covariance matrix (L2ECM) for image representation and its applications. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, volume 7574 of *Lecture Notes in Computer Science*, pages 469--482. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33711-6.

[83] Yuan Li, Chang Huang, and Ram Nevatia. Learning to associate: Hybridboosted multi-target tracker for crowded scene. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2953--2960. IEEE, 2009.

[84] T. Lindeberg. *Scale-space theory in computer vision*. Springer, 1993.

[85] T. Lindeberg. Feature detection with automatic scale selection. *International journal of computer vision*, 30(2): 79--116, 1998.

[86] Meizhu Liu, B.C. Vemuri, S.-I. Amari, and F. Nielsen. Shape retrieval using hierarchical total bregman soft clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(12):2407 --2419, December 2012. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.44.

[87] Yunpeng Liu, Guangwei Li, and Zelin Shi. Covariance Tracking via Geometric Particle Filtering. *EURASIP Journal on Advances in Signal Processing*, 2010:1--10, 2010. ISSN 1687-6172. doi: 10.1155/2010/583918. URL `http://www.hindawi.com/journals/asp/2010/583918.html`.

[88] B.S. lkopf and A.J. Smola. *Learning With Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. Adaptative computation and machine learning series. "The" MIT Press, 2002. ISBN 9780262194754. URL `http://books.google.fr/books?id=y8ORL3DWt4sC`.

[89] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60 (2):91--110, 2004.

[90] B.D. Lucas, T. Kanade, *et al.* An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on Artificial intelligence*, 1981.

[91] James MacQueen *et al.* Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, page 14. California, USA, 1967.

[92] Antoine Manzanera. σ-δ background subtraction and the zipf law. In *Progress in Pattern Recognition, Image Analysis and Applications*, pages 42--51. Springer, 2007.

[93] M.D. McCool, J. Reinders, and A.D. Robison. *Structured Parallel Programming: Patterns for Efficient Computation*. Morgan Kaufmann. Elsevier/Morgan Kaufmann, 2012. ISBN 9780124159938. URL `http://books.google.fr/books?id=zpaHa5cjLwwC`.

[94] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *International journal of computer vision*, 60(1):63--86, 2004.

[95] Anton Milan, Stefan Roth, and Konrad Schindler. Continuous energy minimization for multi-target tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2013.

[96] MINES-ParisTech. PIPS. http://pips4u.org, 1989--2009. Open source, under GPLv3.

[97] H. Moravec. Rover visual obstacle avoidance. In *International Joint Conference on Artificial Intelligence, Vancouver, Canada*, pages 785--790, 1981.

[98] Greg Mori, Xiaofeng Ren, Alexei A Efros, and Jitendra Malik. Recovering human body configurations: Combining segmentation and recognition. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II--326. IEEE, 2004.

[99] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial & Applied Mathematics*, 5(1):32--38, 1957.

[100] F. Nielsen and R. Bhatia. *Matrix Information Geometry*. Springer, 2012. ISBN 9783642302312. URL `http://books.google.fr/books?id=1-QAuwAACAAJ`.

[101] T. Ojala, M. Pietikainen, and D. Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision amp; Image Processing., Proceedings of the 12th IAPR International Conference on*, volume 1, pages 582 --585 vol.1, oct 1994. doi: 10.1109/ICPR.1994.576366.

[102] Timo Ojala, Matti Pietikainen, and Topi Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):971–987, 2002.

[103] Stanley Osher and Nikos Paragios. *Geometric level set methods in imaging, vision, and graphics*. Springer, 2003.

[104] Yanwei Pang, Yuan Yuan, and Xuelong Li. Gabor-based region covariance matrices for face recognition. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(7):989 --993, july 2008. ISSN 1051-8215. doi: 10.1109/TCSVT.2008.924108.

[105] J.R. Parker. *Algorithms for Image Processing and Computer Vision*. Wiley computer publishing. John Wiley & Sons, 1996. ISBN 9780471140566. URL `http://books.google.fr/books?id=7u5RAAAAMAAJ`.

[106] Val'erie Gouet Philippe and Rachid Deriche. Evaluation de détecteurs de points d'intêret pour la couleur evaluation of point of interest detectors for color images, 2000.

[107] M. Pietikäinen. *Computer Vision Using Local Binary Patterns*. Computational Imaging and Vision. Springer London, 2011. ISBN 9780857297488. URL `http://books.google.fr/books?id=wBrZz9FiERsC`.

[108] M. Pietikäinen. *Computer Vision Using Local Binary Patterns*. Computational Imaging and Vision. Springer London, 2011. ISBN 9780857297488. URL `http://books.google.fr/books?id=wBrZz9FiERsC`.

[109] F. Porikli, O. Tuzel, and P. Meer. Covariance tracking using model update based on lie algebra. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 728--735. IEEE, 2006.

[110] Fatih Porikli, Anuj Srivastava, Pavan Turaga, and Ashok Veeraraghavan. Tutorial on differential geometry in computer vision and pattern recognition. CVPR 2102 Tutorial/Short-course June 21, Providence, Rhode Island, 2012.

[111] W.H. Press. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007. ISBN 9780521880688. URL `http://books.google.fr/books?id=1aAOdzK3FegC`.

[112] W. Pusz and S.L. Woronowicz. Functional calculus for sesquilinear forms and the purification map. *Reports on Mathematical Physics*, 8(2):159--170, 1975.

[113] A. Romero, M. Gouiffès, and L. Lacassagne. Feature points tracking adaptive to saturation. In *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*, pages 277 --282, nov. 2011. doi: 10.1109/ICSIPA.2011.6144104.

[114] A. Romero, M. Gouiffés, and Lacassagne L. Covariance descriptor multiple object tracking and re-identi cation with colorspace evaluation. In *Asian Conference on Computer Vision, 2012. ACCV 2012*, 2012.

[115] A. Romero, M. Gouiffès, and L. Lacassagne. Enhanced Local Binary Covariance Matrices ELBCM for texture analysis and object tracking. In *MIRAGE 2013, Berlin, Germany. ACM International Conference Proceedings Series*. Association for Computing Machinery, 2013.

[116] Andrés Romero, M Gouiffès, and L Lacassagne. Total bregman divergence for multiple object tracking. In *IEEE, International Conference on Image Processing (ICIP)*, 2013.

[117] Andrés Romero, Lionel Lacassagne, Ali Hassan Zahraee, and Michèle Gouiffès. Real-time covariance tracking algorithm for embedded systems. In *Design and Architectures for Signal and Image Processing (DASIP), 2013 Conference on*. IEEE, 2013.

[118] David A. Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *Int. J. Comput. Vision*, 77(1-3):125--141, May 2008. ISSN 0920-5691. doi: 10.1007/s11263-007-0075-7. URL `http://dx.doi.org/10.1007/s11263-007-0075-7`.

[119] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1):105--119, 2010.

[120] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of computer vision*, 37(2):151--172, 2000.

[121] W. R. Schwartz and L. S. Davis. Learning Discriminative Appearance-Based Models Using Partial Least Squares. In *Brazilian Symposium on Computer Graphics and Image Processing*, 2009.

[122] Steven A. Shafer. Color. chapter Using color to separate reflection components, pages 43--51. Jones and Bartlett Publishers, Inc., , USA, 1992. ISBN 0-86720-295-5. URL `http://portal.acm.org/citation.cfm?id=136809.136817`.

[123] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

[124] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593--600. IEEE, 1994.

[125] Kevin Smith, Daniel Gatica-perez, Jean marc Odobez, and Sileye Ba. Evaluating multi-object tracking. In *In Workshop on Empirical Evaluation Methods in Computer Vision*, 2005.

[126] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2. IEEE, 1999.

[127] R. Stiefelhagen, K. Bernardin, R. Bowers, J. Garofolo, D. Mostefa, and P. Soundararajan. The CLEAR 2006 evaluation. *Multimodal Technologies for Perception of Humans*, page 1–44, 2007.

[128] J. Stoettinger, A. Hanbury, N. Sebe, and T. Gevers. Do color interest points improve image retrieval? In *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, volume 1, pages I--169--I--172, 16 2007-oct. 19 2007. doi: 10.1109/ICIP.2007.4378918.

[129] K.T. Sturm. Probability measures on metric spaces of nonpositive curvature. *Contemporary mathematics*, 338: 357--390, 2003.

[130] R. Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 2010. ISBN 9781848829350. URL http://books.google.fr/books?id=bXzAlkODwa8C.

[131] Demetri Terzopoulos and Richard Szeliski. Tracking with kalman snakes. *Active vision*, pages 3--20, 1992.

[132] Engin Tola, Vincent Lepetit, and Pascal Fua. DAISY: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Trans. Pattern Anal. Mach. Intell*, 32(5):815--830, 2010. URL http://doi.ieeecomputersociety.org/10.1109/TPAMI.2009.77.

[133] C. Tomasi and T. Kanade. *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ., 1991.

[134] T. Tommasini, A. Fusiello, V. Roberto, and E. Trucco. Robust feature tracking in underwater video sequences. In *OCEANS'98 Conference Proceedings*, volume 1, pages 46--50. IEEE, 1998.

[135] Jing Tou, Yong Tay, and Phooi Lau. Gabor filters as feature images for covariance matrix on texture classification problem. *Advances in Neuro-Information Processing*, page 745–751, 2009.

[136] A. Trémeau, C. Fernandez-Maloigne, and P. Bonton. *Image numérique couleur: De l'acquisition au traitement*. Sciences SUP. Sciences de l'ingénieur. Cours. Dunod, 2004. ISBN 9782100068432. URL http://books.google.fr/books?id=ZvpoHQAACAAJ.

[137] D.N. Truong, Francois Bodin, and A. Seznec. Improving cache behavior of dynamically allocated data structures. In *Parallel Architectures and Compilation Techniques, 1998. Proceedings. 1998 International Conference on*, pages 322--329, 1998. doi: 10.1109/PACT.1998.727268.

[138] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71--86, 1991.

[139] O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. *Computer Vision--ECCV 2006*, pages 589--600, 2006.

[140] O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on riemannian manifolds. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1713--1727, 2008.

[141] Ambrish Tyagi, James W Davis, and Gerasimos Potamianos. Steepest descent for efficient covariance tracking. In *Motion and video Computing, 2008. WMVC 2008. IEEE Workshop on*, page 1–6, 2008.

[142] J. van de Weijer, T. Gevers, and A.W.M. Smeulders. Robust photometric invariant features from the color tensor. *Image Processing, IEEE Transactions on*, 15(1):118 --127, jan. 2006. ISSN 1057-7149. doi: 10.1109/TIP.2005.860343.

[143] B.C. Vemuri, M. Liu, S.I. Amari, and F. Nielsen. Total bregman divergence and its applications to dti analysis. *Medical Imaging, IEEE Transactions on*, 30(2):475--483, 2011.

[144] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2): 137--154, 2004.

[145] Carl Vondrick, Donald Patterson, and Deva Ramanan. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, pages 1--21. ISSN 0920-5691. URL `http://dx.doi.org/10.1007/s11263-012-0564-1`. 10.1007/s11263-012-0564-1.

[146] Andreas Wendel, Sabine Sternig, and Martin Godec. Robustifying the flock of trackers.

[147] N. Wiener. *Cybernetics - 2nd Edition: or the Control and Communication in the Animal and the Machine*. M.I.T. paperback series. MIT Press, 1965. ISBN 9780262730099. URL `http://books.google.fr/books?id=NnM-uISyywAC`.

[148] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Paul Pentland. Pfinder: Real-time tracking of the human body. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):780--785, 1997.

[149] Yi Wu, Bo Wu, Jia Liu, and Hanqing Lu. Probabilistic tracking on riemannian manifolds. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, page 1–4, 2008.

[150] Xu Yan, Xuqing Wu, Ioannis A Kakadiaris, and Shishir K Shah. To track or to detect? an ensemble framework for optimal selection. In *Computer Vision--ECCV 2012*, pages 594--607. Springer, 2012.

[151] J. Yao, J.M. Odobez, *et al.* Fast human detection from videos using covariance features. In *The Eighth International Workshop on Visual Surveillance-VS2008*, 2008.

[152] Jian Yao and J-M Odobez. Multi-layer background subtraction based on color and texture. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1--8. IEEE, 2007.

[153] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm Computing Surveys (CSUR)*, 38 (4):13, 2006.

[154] Ying Zhang and Shutao Li. Gabor-LBP based region covariance descriptor for person re-identification. In *Image and Graphics (ICIG), 2011 Sixth International Conference on*, page 368–371, 2011.