Université de Montréal

Détection de mouvement par modèle biologique de fusion de donnée inspiré de la rétine humaine

par

Sylvain Roux

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

Août, 2015

RÉSUMÉ

Ce mémoire s'intéresse à la détection de mouvement dans une séquence d'images acquises à l'aide d'une caméra fixe. Dans ce problème, la difficulté vient du fait que les mouvements récurrents ou non significatifs de la scène tels que les oscillations d'une branche, l'ombre d'un objet ou les remous d'une surface d'eau doivent être ignorés et classés comme appartenant aux régions statiques de la scène.

La plupart des méthodes de détection de mouvement utilisées à ce jour reposent en fait sur le principe bas-niveau de la modélisation puis la soustraction de l'arrière-plan. Ces méthodes sont simples et rapides mais aussi limitées dans les cas où l'arrière-plan est complexe ou bruité (neige, pluie, ombres, etc.). Cette recherche consiste à proposer une technique d'amélioration de ces algorithmes dont l'idée principale est d'exploiter et mimer deux caractéristiques essentielles du système de vision humain.

Pour assurer une vision nette de l'objet (qu'il soit fixe ou mobile) puis l'analyser et l'identifier, l'œil ne parcourt pas la scène de façon continue, mais opère par une série de "balayages" ou de saccades autour (des points caractéristiques) de l'objet en question. Pour chaque fixation pendant laquelle l'œil reste relativement immobile, l'image est projetée au niveau de la rétine puis interprétée en coordonnées log polaires dont le centre est l'endroit fixé par l'oeil. Les traitements bas-niveau de détection de mouvement doivent donc s'opérer sur cette image transformée qui est centrée pour un point (de vue) particulier de la scène. L'étape suivante (intégration trans-saccadique du Système Visuel Humain (SVH)) consiste ensuite à combiner ces détections de mouvement obtenues pour les différents centres de cette transformée pour fusionner les différentes interprétations visuelles obtenues selon ses différents points de vue.

Mots clés :Détection de mouvement, SVH, saccades, changedetection.net

ABSTRACT

This master thesis revolves around motion detection in sequences recorded from a fixed camera. This situation is challenging since we must ignore insignificant recurring motions such as oscillating branches, shadows, or waves on the surface of the water. Those must be classified as belonging to the background and static.

Most motion detection techniques used nowadays are based on the simple and low level principle of background modeling and subtraction. These techniques are simple and fast but they reach their limit when they have to deal with complex or noisy images (from snowy, rainy, sunny weather, etc.). This research consists of proposing a technique aiming to improve those algorithms by mimicking two essential characteristics of the Human Visual System (HVS).

To obtain a clear vision of an object (static or mobile) and then to analyse and identify it, our eye doesn't analyse the scene continuously but operates through several sweeping motions, or saccades, across the object. During each moment when the eye stays fixed, the image is projected on the retina and then interpreted as described by log-polar coordinates, where the center is the point fixed by the eye. Low level detection treatment should then operate on this transformed image which is centered on a particular point of view of the scene. The second step (the trans-saccadic integration of the HVS) is to combine all those data gathered from different points of view.

Keywords : Image processing, motion detection, change detection, background detection, HVS, saccades, image corticale, changedetection.net

TABLE DES MATIÈRES

LISTE DES TABLES

LISTE DES FIGURES

LISTE DES ALGORITHMES

ABREVIATIONS

SVH	Système Visuel Humain
GMM	Gaussian Mixture Model (Modèle de Mèlange de Gaussiennes)
RVB	Rouge Vert Bleu
CDNET	changedetection.net

REMERCIEMENTS

Je ne saurais, réellement, trouver les expressions éloquentes que mérite mon directeur Max Mignotte, professeur agrégé de l'Université de Montréal, pour son aide, sa disponibilité, sa confiance, et sa patience. Je remercie l'équipe enseignante du DIRO pour l'ouverture aux connaissances qui m'y a été offerte. Je tiens aussi à remercier Pierre-Marc Jodoin et Nil Goyette de l'Université de Sherbrooke pour leurs précieuses informations. Je tiens aussi à remercier les membres du jury. Mes remerciements vont enfin à mes parents, mon frère ainsi que mes amis dont le soutient a été important.

Chapitre 1

INTRODUCTION GÉNÉRALE

1.1 Problématique

Les performances de la capacité de traitement de l'information du système visuel humain (SVH) sont de loin bien supérieures aux algorithmes de traitement d'images existants. Cette caractéristique vient du fait que les informations transmises par l'oeil au cortex ne sont pas les images brutes acquises par les récepteurs sensoriels de l'oeil et que le SVH réalise en fait différents prétraitements des données au niveau de la rétine et du cortex qui vont ensuite faciliter l'interprétation des données. Lorsqu'une scène s'expose à nous, nous ne voyons donc pas objectivement les choses et notre vision ne se limite pas à une mesure de la quantité et de la longueur d'onde des photons provenant de la scène ; nous interprétons l'information reçue par l'œil grâce à notre cerveau, grâce à nos connaissances, à nos concepts : dans ce que l'on regarde, on va chercher à comprendre ce qui se passe. La vision, comme les autres sens, concerne donc la perception et l'interprétation du réel, on parle de vision par ordinateur lorsque l'on arrive à douer celui-ci d'outils pour interpréter une vidéo à la façon d'un humain, soit en déduire des nouvelles informations, autres que celles constituant les données brutes de la vidéo.

L'œil humain est un organe complexe. La détection de ce qui est en mouvement a sûrement été un point crucial dans son évolution dans le sens où celle-ci permet de détecter prédateur ou proie, et donc de survivre. Elle est aussi un point crucial du traitement d'images de vidéo à bas niveau puisqu'elle permet à des algorithmes de plus haut niveau d'analyser ces mouvements. Détecter les mouvements nous permet

d'isoler les éléments qui vont offrir le plus d'interaction avec l'environnement. Grâce aux techniques de traitement d'images, on cherche à douer l'ordinateur de la capacité à détecter avec précision le mouvement, et de copier ainsi le rôle du cerveau. Cependant la méthode de capture et d'enregistrement de l'image diffère de beaucoup de celle utilisée par notre SVH. L'image informatique est représentée par un ensemble de pixels organisés sur un repère cartésien discret alors que l'œil reçoit la lumière sur la rétine, composée de capteurs distribués autour de son centre en une formation décrivant un système de coordonnés polaires. Il s'agit dans cette étude de comprendre l'intérêt qu'à la forme de l'œil dans la vision par rapport à une matrice de pixel, et éventuellement en tirer parti pour proposer une amélioration des techniques de vision par ordinateur.

1.2 État de l'art

Le cas qui nous intéresse dans ce mémoire concerne le problème de la détection de mouvement dans une séquence d'images acquises à l'aide d'une caméra fixe (ou de vidéo-surveillance). Ce cas de figure est depuis longtemps connu et étudié par la communauté scientifique et de nombreux algorithmes de détection de mouvements différents existent déjà. L'approche informatique usuelle de détection du mouvement va produire pour chaque image d'une vidéo une image binaire en noir et blanc (cf. Fig. 1.1 à droite) classant respectivement chaque pixel comme appartenant aux régions statiques (noires) ou mobiles (blanches) de la scène.

(a) **(b)**

FIGURE 1.1: *(a) Image numéro 2000 de la séquence boats de changedetection.net. (b) le résultat de la détection de mouvement à droite grâce à l'algorithme GMM de Z. Zivkovic [1] associé à un filtre médian de* $5 * 5$ *pixels.*

1.2.1 Un algorithme de base pour la détection de mouvements

Zoran Zivkovic [1] propose une approche de détection du mouvement au niveau du pixel utilisant un modèle de densité de mélange [1] offrant une plus grande souplesse que des modèles plus rudimentaires (tels qu'utilisants seulement un seuillage sur les gradients temporels, cf. chapitre 2). En effet, pour une vidéo analysée, chaque pixel de chaque image se réfèrera à un modèle à plusieurs paramètres, évoluant dans le temps en fonction des nouvelles images, permettant d'estimer s'il appartient au fond (statique) ou non (mobile).

Pour imiter les caractéristiques du SVH, nous allons utiliser la représentation d'une image selon un repère logarithmique plutôt qu'un repère cartésien pour étudier les probables gains que l'on pourrait obtenir dans un problème de détection du mouvement. Ainsi une approche telle que celle proposée par Zoran Zivkovic, comme base de détection de mouvement, paraît pertinente pour mettre en évidence les éventuels

1. En anglais : GMM, gaussian mixture model.

intérêts de cette transformation d'autant plus que cette méthode est à la fois de bas niveau et rapide mais présente aussi des points à améliorer. Les points faibles de cette méthode sont illustrés dans la Fig. 1.1, dans laquelle on peut remarquer quelques erreurs de détection et plus précisément des faux négatifs (en noir) sur la coque du bateau en mouvement, ou des faux positifs au niveau des vagues [2].

1.2.2 ChangeDetection.net CDNET

Le site Web changedetection.net propose une base de données pour évaluer objectivement les algorithmes de détection de changement [3]". Une mesure de succès "F-mesure" allant de 0 à 1 est déterminée à travers un processus détaillé dans le chapitre 6. Ce site Web contient deux ensembles de données, la plus récente de 2014 propose 11 catégories regroupant en moyenne 5 vidéos par catégorie ou chaque catégorie représente un défi particulier pour le problème de la détection de mouvement (bruit, ombre, pénombre, reflets, changement d'illumination, mouvements non significatifs, etc.).

Le site héberge les résultats de nombreux algorithmes différents sur cette base de données, notamment celui de Z. Zivkovic utilisant le modèle GMM. La mesure et l'étude des résultats de ce projet seront donc plus aisées dans le sens où CDNET nous donne une référence pour calculer les gains apportés par ce projet.

2. Les résultats de détection étant binaires, ils peuvent être positifs, négatifs, ou faussement détecté comme positif ou négatif, du fait que nous traitons ici d'un arrière plan dynamique. (cf. Chapitre 6).

3. Les termes détection de changement, mouvement, et fond sont utilisés indifféremment dans la suite.

1.2.3 Utilisations antérieures du système de coordonnées log polaire

Le système de coordonnées log-polaire décrivant l'implantation des cellules captant la lumière dans la fovéa [4], par sa résolution spatiale plus grande au centre et plus éparse en périphérie, a déjà suscité un intérêt de la communauté scientifique dont une partie de l'étendue sera étudiée chapitre 4. L'article de Max Mignotte [6] présente un algorithme simple et efficace de détection de contour pour image. Cet article propose un modèle s'approchant du fonctionnement de l'œil à travers des transformations log-polaire de l'image en différents centres de manière à mimer le mouvement saccadique de l'œil, ce dont ce projet tirera parti. Plus généralement, l'article [10] de C. F.R Weiman et G. Chaikin met en avant le fait que la transformation log-polaire permet la simplification de traitements spatiaux sur l'image.

1.3 Conclusion

Nous poursuivrons cet état de l'art dans le Chapitre 2 en nous concentrant sur le travail sur lequel se base l'algorithme de Zoran Zivkovic [1]. Nous passerons en revue des méthodes de détection de mouvement par modélisation de l'arrière plan en allant dans un sens croissant de complexité, pour finalement détailler le fonctionnement de l'algorithme [1] qui nous intéresse le plus.

4. Zone de la rétine dans laquelle se concentre la majorité des cellules réceptrices à la lumière dans l'œil.

Chapitre 2

MÉTHODES DE DÉTECTION DE MOUVEMENT PAR MODÉLISATION DE L'ARRIÈRE-PLAN

2.1 Introduction

La détection de mouvement par modélisation puis soustraction de l'arrière-plan (aussi connue sous les termes anglais de *background subtraction* ou *foreground detection*) permet de dissocier dans une séquence d'images les éléments ou régions en mouvement (ou mobiles) de ceux appartenant au *décor* de la scène (immobiles ou statiques). Les techniques de détection de mouvement utilisant cette stratégie se reposent sur l'hypothèse principale que la séquence vidéo, prise par une caméra fixe, est composée d'un *background* ou arrière-plan statique au-devant duquel les objets en mouvement peuvent être ainsi facilement détectés. La détection du mouvement est fondamentalement une opération de traitement d'images bas niveau qui est ensuite cruciale pour l'élaboration et la conception de techniques plus évoluées en vision par ordinateur telles que le suivi d'objets mobiles, l'analyse du mouvement humain, la reconnaissance d'actions humaines, la compression ou la compréhension de la scène.

Une méthode simple de détection d'arrière-plan consiste à prendre une image de référence de la scène ne contenant aucun objet mobile puis, par simple différence et seuillage avec les différentes images de la séquence vidéo, identifier ensuite les pixels appartenant aux éléments mobiles n'appartenant pas au *décor* (statique) de la scène. Cependant on comprend vite que l'usage d'une simple image de référence, décrivant l'arrière-plan statique, constitue un modèle trop simple et peu fiable en réalité. En effet, outre le fait qu'il est requis que la caméra soit parfaitement fixe, la luminosité

de la scène change et varie dans le temps, l'image peut être entachée de bruit, un mouvement récurrent et inintéressant tel que celui d'une branche dans le vent qui peut être détectée en faux positif (tout comme la neige ou la réflexion soudaine du soleil sur une vitre peut engendrer une fausse détection), le support de la caméra peut trembler, etc. Ces erreurs possibles ont incité les auteurs à proposer des modèles de modélisation d'arrière-plan plus élaborés.

2.2 Modélisation d'arrière-plan par simple image

Les méthodes présentées dans cette section se basent sur un modèle d'arrière-plan constitué d'une simple image qui est ensuite exploitée pour détecter les régions en mouvement existant dans la séquence par simple opération de soustraction et seuillage à l'image courante.

2.2.1 Différence

Comme il a été dit en introduction, l'approche la plus évidente consiste à prendre, comme modèle d'arrière-plan, une des images de la séquence ne contenant pas d'objet mobile. Notons I, l'ensemble des images de la séquence, $\mathbf{I}(t)$, une image de cette séquence à un instant t et $\mathbf{I}(x, y, t)$ ou $\mathbf{I}(\mathbf{s}, t)$, un pixel composant l'image de cette séquence à un instant t et localisé en $\mathbf{s} = (x, y)$. Ainsi on compare au modèle de fond, noté BG, l'ensemble des images de la séquence :

$$|\mathbf{I}(\mathbf{s}, t) - \mathrm{BG}(\mathbf{s}, t)| > th \tag{2.1}$$

En appliquant un seuil th, on obtient une séquence d'images contenant les pixels appartenant au premier plan ou aux régions mobiles de la scène. Cependant l'arrière-plan étant représenté ici par une image unique, ce modèle est limité par sa simplicité et ne peut refléter le *dynamisme* ou la variation d'intensité locale que possède réellement un arrière-plan dans une vraie séquence d'images ; comme l'oscillation périodique ou

aléatoire d'une branche d'arbre dans le vent cachant ou non la lumière du soleil et créant alternativement une ombre ou une réflexion de cette lumière sur le sol.

Une façon de résoudre ce problème est expliquée en [10] et consiste à considérer simplement l'image précédente, dans la séquence, comme image d'arrière-plan pour l'image courante :

$$|\mathbf{I}(\mathbf{s},t) - |\mathbf{I}(\mathbf{s},t-1)| > th \tag{2.2}$$

Mais cette stratégie rend la méthode de détection de mouvement peu fiable notamment dans le cas où la vitesse des régions mobiles est trop lente ou dans le cas d'objets mobiles peu texturés possédant des régions d'intensité uniforme et pour lesquelles aucune variation d'intensité est visible lorsque l'objet *glisse* sur lui-même.

2.2.2 Filtre moyen

Cette troisième méthode [10] utilise comme modèle d'arrière-plan la moyenne des n images précédant l'image courante :

$$\mathrm{BG}(\mathbf{s},t) = \frac{1}{n}\sum_{k=0}^{n-1}\mathbf{I}(\mathbf{s},t-k) \quad \text{et} \quad |\mathbf{I}(\mathbf{s},t) - \mathrm{BG}(\mathbf{s},t)| > th \tag{2.3}$$

Cependant cette méthode montre vite des faiblesses lors de passages récurrents d'objets mobiles qui vont finir par laisser une trace dans le modèle d'arrière-plan même si le paramètre n est estimé adaptivement en fonction du nombre d'images par seconde de la vidéo et de la vitesse des objets mobiles.

2.2.3 Filtre median

Dans le cas où chaque pixel de la séquence d'images n'est pas majoritairement caché par un objet mobile, on peut aussi utiliser un filtre médian temporel [10] calculé pour chaque pixel et utilisant les précédentes images de la séquence pour estimer un modèle d'arrière-plan. Cette stratégie va permettre d'obtenir un modèle de fond d'une

meilleure qualité qu'une moyenne temporelle, permettant d'éviter les traces d'objets mobiles sur le modèle d'arrière-plan :

$$\mathrm{BG}(\mathbf{s},t) = \mathrm{median}_{k=0}^{n-1}\{\mathbf{I}(\mathbf{s},t-k)\} \quad \text{et} \quad |\mathbf{I}(\mathbf{s},t) - \mathrm{BG}(\mathbf{s},t)| > th \tag{2.4}$$

Comme nous l'avons déjà noté, ces méthodes de détection reposent toutes sur le même principe et souffrent de deux désavantages majeurs : l'arrière-plan est une image unique (bien que parfois issu de la fusion ou la moyenne de plusieurs stratégies), le seuil th est non dynamique par rapport au temps et reste aussi le même pour l'ensemble des pixels. Cette absence d'adaptativité spatiale et temporelle rend ces méthodes peu robustes face aux changements de luminosité et aux mouvements récurrents non significatifs appartenant à l'arrière-plan.

Cependant ces approches simples (donc facile à mettre en oeuvre et rapide) ne sont pas à exclure et peuvent être utilisées efficacement dans des cas particuliers où les conditions d'illumination sont connues et/ou le décor de la scène se distingue facilement des objets mobiles. Pour des scènes plus complexes, par exemple de plein air, des méthodes plus élaborées reposant sur la définition d'un modèle de mélange de densités pour chaque pixel de l'arrière-plan doivent être considérées.

2.3 Mélange de densités pour la modélisation de l'arrière-plan

2.3.1 Modèle de mélange de densités proposé par Stauffer et al.

C. Stauffer et W.E.L. Grimson ont proposé dans leur article [11] une nouvelle approche pour la modélisation de l'arrière-plan qui se distingue des méthodes précédentes où celui-ci est modélisé par une simple image qui est ensuite comparée, par simple seuillage, aux autres images de la séquence. Dans leur stratégie, chaque pixel de l'arrière-plan est modélisé par un mélange de gaussiennes. La gaussienne a été choisie car cette distribution modélise bien les variations temporelles d'intensités prises

par chaque pixel d'un décor ou d'un arrière-plan *dynamique*. En effet, dans le cas ou un arrière-plan doit tenir compte localement de la possible oscillation périodique ou aléatoire d'une branche d'arbre dans le vent cachant ou non la lumière du soleil, la modélisation par un mélange d'au moins deux gaussiennes prend tout son sens ; une des deux distributions gaussienne modélisera la variation temporelle d'intensité du pixel lorsque la branche d'arbre obstrue le soleil et une autre représentera les différentes intensités émises localement par le soleil. Au cours de l'analyse de la vidéo, l'intensité de chaque pixel est ensuite comparée à ce mélange de gaussiennes, représentant une distribution de probabilité des intensités possibles appartenant au modèle d'arrière-plan dynamique. Une probabilité trop basse d'appartenir à ce mélange de gaussiennes indiquera que le pixel appartient à une région mobile (premier plan).

2.3.2 Modèle de mélange [11]

Pour chaque pixel $\mathbf{s}$ de l'image, à un temps t, la distribution des intensités possibles caractérisant l'arrière-plan *dynamique* ($\mathrm{BG}(\mathbf{s},t)$) et le premier plan ($\mathrm{FG}(\mathbf{s},t)$) est calculée à partir de l'ensemble des niveaux de gris (ou de couleurs) données par les T dernières images de la séquence : i.e., $\{\mathbf{I}(\mathbf{s},t),\ldots,\mathbf{I}(\mathbf{s},t-T)\}$ que l'on suppose être distribuées selon un mélange de K gaussiennes dont on estimera plus tard les paramètres :

$$P\big(\mathrm{BG}(s,t)+\mathrm{FG}(\mathbf{s},t)\big)=\sum_{k=1}^{K}\hat{\omega}_{k,t}\cdot\mathcal{N}(\mathbf{I}(\mathbf{s},t),\hat{\mu}_{k,t},\hat{\Sigma}_{k,t}) \tag{2.5}$$

Les variables $\hat{\omega}_{k,t}$, $\hat{\mu}_{k,t}$, et $\hat{\sigma}^2_{k,t}$, représentent respectivement une estimation de la proportion, de la moyenne et de la variance de la k-ième gaussienne à l'instant t. Id est la matrice identité. On pose $\hat{\Sigma}_{k,t}=\hat{\sigma}^2_{k,t}\,Id$, ce qui suppose ici que les trois canaux de couleur sont indépendants et de même variance. $\mathcal{N}$ est la densité de probabilité

normale (de dimension trois pour des niveaux de couleurs) définie par :

$$\mathcal{N}(\mathbf{I}(\mathbf{s},t),\hat{\mu}_{k,t},\hat{\Sigma}_{k,t}) = \frac{1}{(2\pi)^{\frac{n}{2}}\left|\hat{\Sigma}_{k,t}^{\frac{1}{2}}\right|}\exp\{-\frac{1}{2}(\mathbf{I}(\mathbf{s},t)-\hat{\mu}_{k,t})^T\hat{\Sigma}_{k,t}^{-1}(\mathbf{I}(\mathbf{s},t)-\hat{\mu}_{k,t})\} \quad (2.6)$$

L'intensite $\mathbf{I}(\mathbf{s},t)$ de chaque pixel est ensuite comparee a ce melange de gaussiennes. Dans le cas où ce pixel n'appartient à aucune des K distributions (pour une valeur de couleur au-delà de 2.5σ de chacune des K distributions), la moins probable des distributions de notre mélange est remplacée par une distribution avec $\hat{\mu_{k,t}} = \mathbf{I}(\mathbf{s},t)$, une grande variance $\hat{\sigma}^2$ et une faible proportion. Dans le cas où ce pixel appartient à ce modèle de densité, l'ensemble des paramètres $\hat{\omega}_{k,t}$, $\hat{\mu}_{k,t}$, et $\hat{\sigma}^2_{k,t}$ du mélange de distributions est ensuite ajusté par une technique proche de l'algorithme des K-moyennes par les relations suivantes :

$$\begin{aligned}
\hat{\omega}_{k,t} &\leftarrow \hat{\omega}_{k,t-1} + \alpha(o_{k,t} - \hat{\omega}_{k,t-1}) \\
\hat{\mu}_{k,t} &\leftarrow \hat{\mu}_{k,t-1} + o_{k,t}\,\rho(\mathbf{I}(\mathbf{s},t) - \hat{\mu}_{k,t-1}) \\
\hat{\sigma}^2_{k,t} &\leftarrow \hat{\sigma}^2_{k,t-1} + o_{k,t}\,\rho((\mathbf{I}(\mathbf{s},t) - \hat{\mu}_{k,t-1})^T(\mathbf{I}(\mathbf{s},t) - \hat{\mu}_{k,t-1}) - \hat{\sigma}^2_{k,t-1})
\end{aligned} \quad (2.7)$$

La variable $o_{k,t}$ prend pour valeur 1 pour la gaussienne k du mélange pour laquelle l'échantillon $\mathbf{I}(\mathbf{s},t)$ est le plus probable et $o_{k,t} = 0$ pour les autres. De cette façon, la variance et la moyenne changent uniquement pour la gaussienne retenue (et les proportions du mélange sont normalisées pour que leur somme soit égale à un). Le paramètre α est un facteur d'oubli constant qui permet de réduire l'importance des échantillons passés par rapport aux nouveaux échantillons ($\alpha \approx 1/T$). ρ est un second facteur d'oubli défini ici par la relation $\rho = \alpha\,\mathcal{N}(\hat{\mu}_{k,t},\Sigma_{k,t})$. Ainsi les gaussiennes du mélange représentent l'arrière plan en prenant compte des changements qui peuvent advenir dans la scène au fil du temps.

On peut maintenant approximer le modèle de l'arrière-plan ($\mathrm{BG}(\mathbf{s},t)$) en retenant seulement les $\hat{b}$ premières gaussiennes de notre mélange triées selon les valeurs de

$\hat{\omega}_{k,t}/\hat{\sigma}_{k,t}$ croissantes :

$$\hat{b} = \arg\min_{b} \left(\sum_{k=1}^{b} \hat{\omega}_{k,t} > (1 - c_f) \right) \tag{2.8}$$

Ainsi, les gaussiennes qui ont les variances les plus étroites et les proportions les plus grandes sont celles qui seront majoritairement sélectionnées comme faisant partie du modèle de l'arrière-plan. Inversement, les éléments en mouvement introduits dans la scène seront représentés dans le modèle par des gaussiennes de faibles poids et de fortes variances. Le paramètre c_f est défini comme étant la proportion maximum des données dans le modèle jugée comme n'appartenant pas à l'arrière-plan. Ainsi si un nouvel échantillon $\mathbf{I}(\mathbf{s}, t)$ est associé à une des B premières distributions, alors il sera considéré appartenant à l'arrière-plan. Dans le cas contraire, il appartiendra au premier plan.

2.3.3 Une amélioration de GMM

Dans son article [1], Z. Zivkovic se base sur les travaux [11] de Stauffer et Grimson pour apporter différentes améliorations au modèle précédent. L'idée est de rendre plus adaptatif les paramètres et le nombre de gaussiennes associé à chaque pixel de l'arrière-plan. L'estimation des moyennes et des variances et le calcul de la proportion des gaussiennes est ajustée par Zivkovic de la façon suivante :

$$\begin{aligned}
\hat{\omega}_{k,t} &\leftarrow \hat{\omega}_{k,t-1} + \alpha\,(o_{k,t} - \hat{\omega}_{k,t-1}) - \alpha\, c_T \\
\hat{\mu}_{k,t} &\leftarrow \hat{\mu}_{k,t-1} + o_{k,t}\,\frac{\alpha}{\hat{\omega}_{k,t}}(\mathbf{I}(\mathbf{s},t) - \hat{\mu}_{k,t-1}) \\
\hat{\sigma}^2_{k,t} &\leftarrow \hat{\sigma}^2_{k,t-1} + o_{k,t}\,\frac{\alpha}{\hat{\omega}_{k,t}}\Big((\mathbf{I}(\mathbf{s},t) - \hat{\mu}_{k,t-1})^T(\mathbf{I}(\mathbf{s},t) - \hat{\mu}_{k,t-1}) - \hat{\sigma}^2_{k,t-1}\Big)
\end{aligned} \tag{2.10}$$

Le nouveau calcul du poids des gaussiennes est ajusté pour chaque échantillon $\mathbf{I}(\mathbf{s},t)$ par un calcul probabiliste [12] en fonction des résultats précédents. Dans le calcul du poids $\hat{\omega}_{k,t}$, le paramètre c_T présente un critère de probabilité d'existence dans le modèle de la gaussienne k associée au nouvel échantillon (si le poids d'une gaussienne devient négatif, alors cette composante est retirée du modèle). De plus, le calcul de la distance d'un échantillon $\mathbf{I}(\mathbf{s},t)$ par rapport à une gaussienne k est celui de la distance D de Mahalanobis :

$$D_k(\mathbf{I}(\mathbf{s},t)) = \sqrt{(\mathbf{I}(\mathbf{s},t) - \hat{\mu}_{k,t-1})^T(\mathbf{I}(\mathbf{s},t) - \hat{\mu}_{k,t-1})/\hat{\sigma}^2_{k,t-1}} \tag{2.11}$$

Cette distance est évaluée en terme du nombre d'écart-types la séparant d'une gaussienne du mélange de l'arrière-plan qui est représentée, comme dans le modèle précédent, par les $\hat{b}$ premières gaussiennes en terme de valeurs de $\hat{\omega}_{k,t}/\hat{\sigma}_{k,t}$ croissantes. De même, les régions mobiles sont aussi représentées par des gaussiennes de faible poids.

2.4 Conclusion

L'algorithme GMM de Zivkovik, du fait de son traitement bas niveau de l'image (chaque pixel est traité individuellement des autres) et de sa capacité d'adaptation aux différentes situations rencontrées dans les séquences vidéo, se présente comme un bon candidat pour notre étude. Sa simplicité nous permettra de mettre en avant l'avantage d'intégrer une technique de vision imitant les propriété de la vision humaine, *via* la fusion de traitement (bas niveau) sur des transformations log-polaire de la scène, pour un problème de détection de mouvement.

Chapitre 3

APPORTS DU SVH DANS LA DÉTECTION DE MOUVEMENTS

3.1 SVH — L'anatomie de l'oeil

La lumière qui entre dans l'oeil, se recompose sur la rétine par le système optique formée de la cornée, de l'iris et la pupille, puis est transformée en influx nerveux, interprétable par le cerveau. Pour ce faire, l'image projetée sur la surface de la rétine est absorbée par les millions de cellules photo-réceptrices la composant. Ces cellules se répartissent en deux catégories ; les bâtonnets et les cônes. Les premiers captent l'intensité lumineuse et retournent cette unique information au cerveau, ils sont principalement sensibles à des longueurs d'onde moyennes proche du vert (cf. Fig. 3.1). Les seconds sont les cônes, ils détectent individuellement selon leur type les longueurs d'onde du bleu, du vert et du rouge (cf. Fig. 3.1), et fournissent ainsi une information sur la couleur de l'image.

Les cônes se concentrent sur la fovéa (cf. Fig. 3.2) et en dehors de celle-ci, leur densité diminue drastiquement au profit de celle des bâtonnets. D'un point de vue général, on peut retenir que la distribution des photorécepteurs est plus dense au centre, et va en diminuant vers l'extérieur de la surface de la rétine. Il est à noter que la perte de densité des photorécepteurs sur l'extérieur de la surface de la rétine va de pair avec une augmentation de leur surface individuelle. La vision est donc plus précise au centre de l'oeil qu'en périphérie.

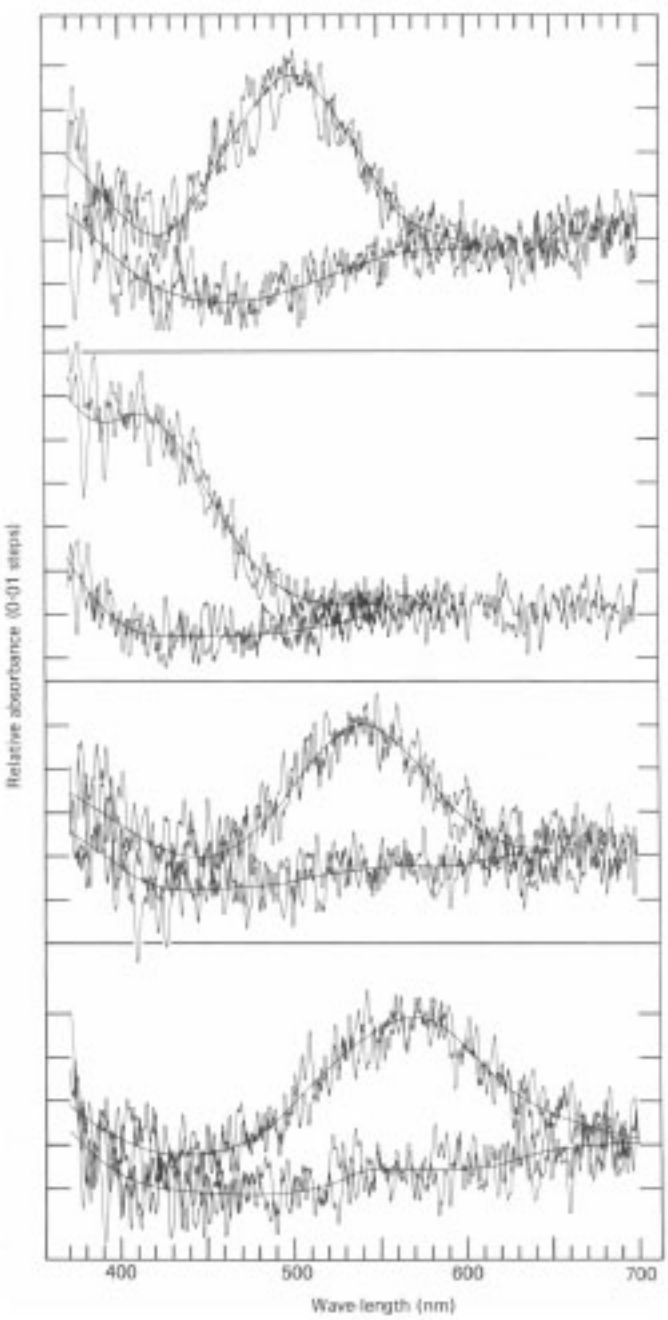

FIGURE 3.1: *Enregistrement micro-spectrophotométrique d'un bâtonnet (premier graphique), et successivement des cônes bleu, vert, et rouge. La courbe supérieure montre en abscisse l'absorbance par rapport à la longueur d'onde de la lumière observée sur le photorécepteur concerné tandis que celle du bas est celle d'une surface témoin [2] .*

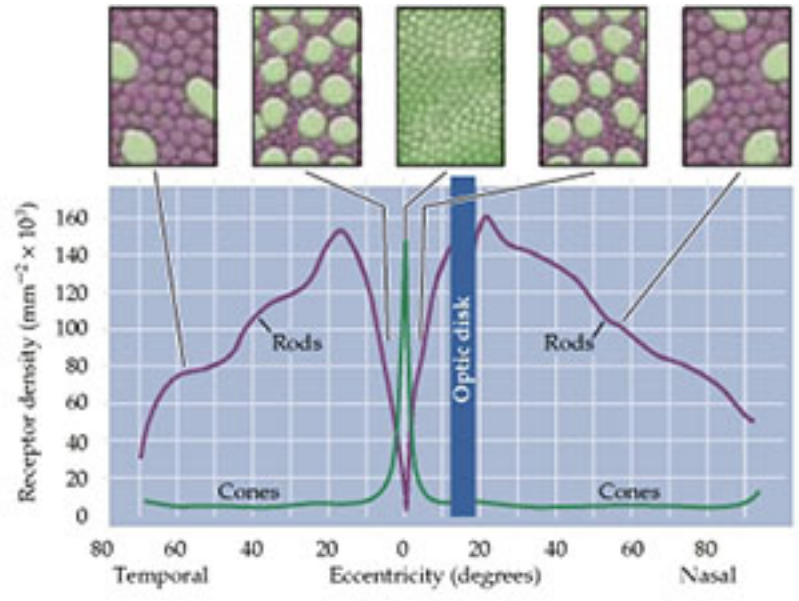

FIGURE 3.2: *Distribution des cônes et des bâtonnets sur la rétine humaine [3].*

La présence de trois récepteurs pour des longueurs d'onde du rouge, du vert et du bleu révèle que ces trois couleurs sont suffisantes pour décrire le spectre de la lumière visible par l'homme. L'organisation des cônes et des bâtonnets révèle que nous sommes principalement sensibles au changement d'intensité sur notre vision périphérique mais peu au changement de couleurs, ce qui s'inverse sur la fovéa.

3.2 La vidéo numérique

Une vidéo se définit par une suite d'images projetées successivement selon une fréquence temporelle, créant ainsi une illusion d'animation de la scène. Comme l'oeil, une caméra va concentrer à travers une lentille la lumière qui lui provient d'une scène sur une surface photosensible. Cependant la capture vidéo implique en plus de stocker l'image captée.

3.2.1 Le stockage numérique de la vidéo

À l'écran, l'image est représentée par une matrice de pixels. En variant les couleurs qui les composent, les pixels étant suffisamment fins, donnent, à une certaine distance, l'illusion d'apparaître comme une unique image. Sur un écran LCD standard, pour représenter l'image, les pixels disposent de trois cellules de fréquences différentes variables : rouge, vert et bleu. De la même manière que chaque cône de la rétine perçoit indépendamment une de ces longueurs d'onde pour les regrouper en une seule couleur interprétable par notre cerveau, la combinaison de ces trois cellules en variant leur intensité permet de recomposer le spectre de la lumière visible. Cependant les limites de niveaux d'intensité produits par les cellules du pixel limitent le nombre de couleurs affichables. Ainsi on comprend que les valeurs de ces pixels sont stockées numériquement pour afficher l'image, selon diverses méthodes.

L'enregistrement de l'image peut donc se faire en stockant naïvement les données des pixels représentant l'image, les unes après les autres dans la mémoire. Chaque pixel sera représenté par trois octets, chacun représentant un canal de couleur du système RVB [1]. On a ainsi une image volumineuse en terme de mémoire, mais qui peut être compressée par la suite grâce à des algorithmes tels que JPEG, au détriment de la qualité de l'image, perceptibles ou non.

3.2.2 Capture de la vidéo

Le dispositif anatomique qui permet à la rétine de l'oeil de capter la lumière pour la transmettre ensuite au cerveau, est modélisé pour la caméra par un capteur photographique. En rappelant le fonctionnement de l'écran, ce capteur se compose d'une matrice de photorécepteurs individuellement sensibles aux longueurs d'onde du rouge, du vert et du bleu, tel que le système RVB. Ainsi on enregistre l'empreinte numérique de l'image projetée fournie par les récepteurs.

1. Rouge Vert Bleu, Red Green Blue RGB en anglais

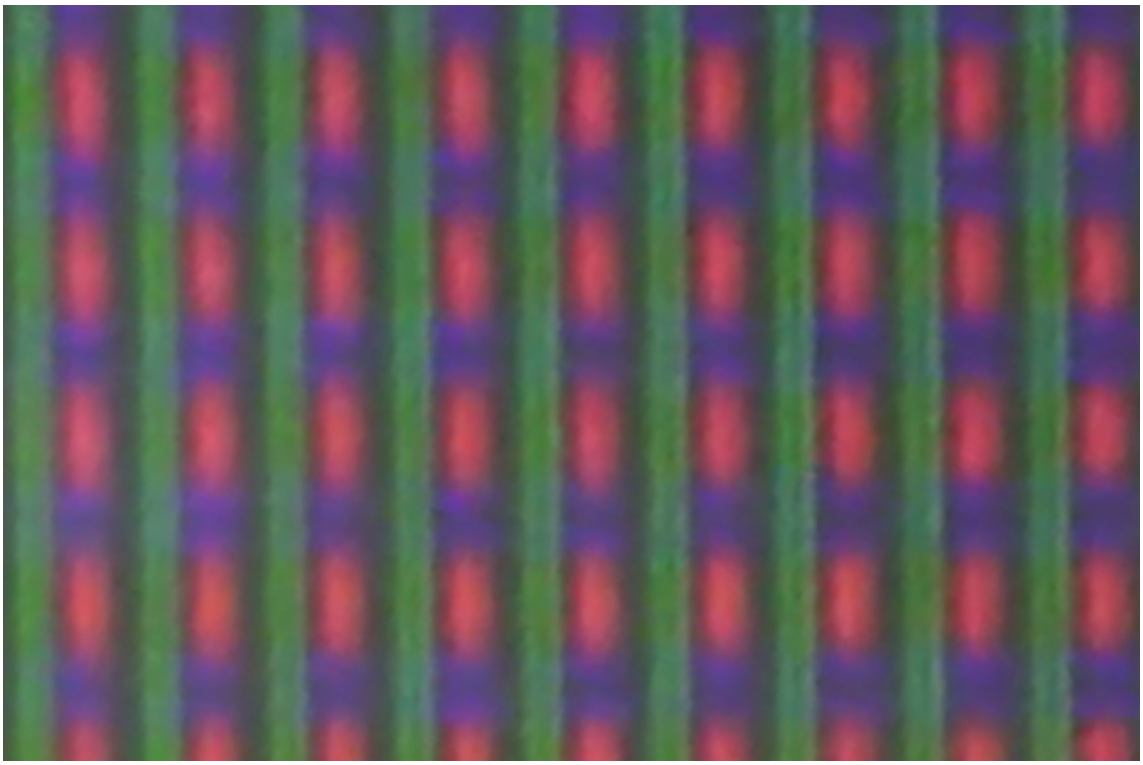

FIGURE 3.3: *Capteur photographique CCD photographié à travers un microscope*[2]*. L'organisation de celui-ci décrit un filtre de Bayer, du nom de son inventeur, Bryce Bayer [22].*

On remarque sur le capteur photographique (cf. Fig. 3.3) une quantité de pixel verts deux fois supérieures au nombre de pixels bleus ou rouges. Comme expliqué en début de chapitre, les bâtonnets de la rétine ne fournissent qu'une information d'intensité, mais sont par nature principalement réactifs aux lumières de longueur d'onde moyenne (cf. Fig. 3.1), soit vertes. Le capteur photographique n'ayant pas de photorécepteurs dédiés à l'intensité, mais des capteurs récupérant à la fois l'intensité et la longueur d'onde, inclue deux fois plus de capteurs verts que de bleus ou de rouges, copiant ainsi le fonctionnement de l'oeil.

2. Crédits : utilisateur wikipedia Binarysequence

3.3 Modélisation de la rétine par coordonnées log-polaire

La sensibilité des récepteurs photosensible d'un capteur de caméra est homogène et, de ce fait, l'image y est enregistrée de la même manière, indifféremment de l'endroit où se trouve le capteur. Contrairement à un capteur photosensible numérique, l'organisation de la rétine permet au SVH (système visuel humain) une forte acuité visuelle au centre de la vue [3], au détriment de la périphérie. Ces variations de densité et de taille des photo-récepteurs sur la rétine (cf. Fig. 3.2) fournissent au cerveau une image transformée par rapport à celle d'un capteur photographique. Cette transformation topologique réalisée par le SVH peut être modélisée par une transformée log-polaire de l'image projetée sur la rétine. Une structure d'échantillonnage adaptée (cf. Fig. 3.4) appliquée à l'image cartésienne en un centre donné permet de réorganiser sur un repère les données de l'image à transformer.

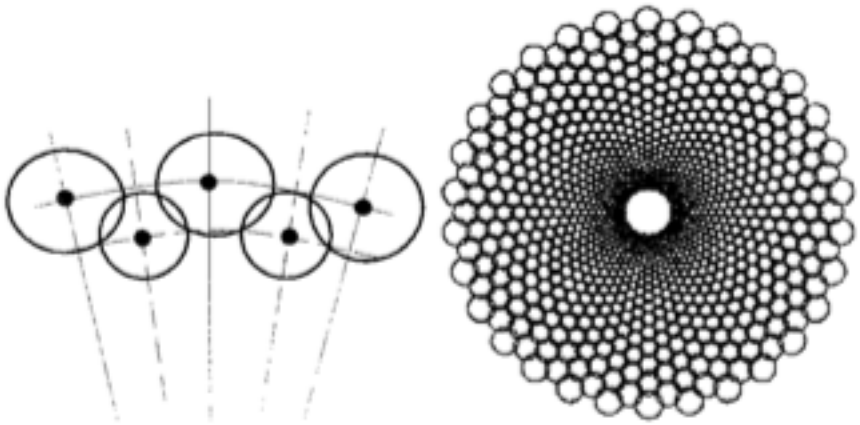

FIGURE 3.4: *Système de coordonnées log-polaire à 60 angles [5].*

La projection corticale réalise un filtre passe-bas qui atténue d'autant plus les détails de l'image (hautes fréquences) que ceux-ci se trouvent vers l'extérieure de la fovéa. Par contre, cette transformée log polaire de l'image conserve les détails exis-

tant au centre de la rétine simulée (ou l'origine de la transformation). En s'éloignant du centre de cette transformation, la vision périphérique du SVH s'affiche de plus en plus compressée (cf. Fig. 3.5). En prenant en compte cette notion de vision centrale (précise) et périphérique (compressée), on peut ainsi proposer intelligemment un modèle numérique de traitement d'images imitant biologiquement le SVH.

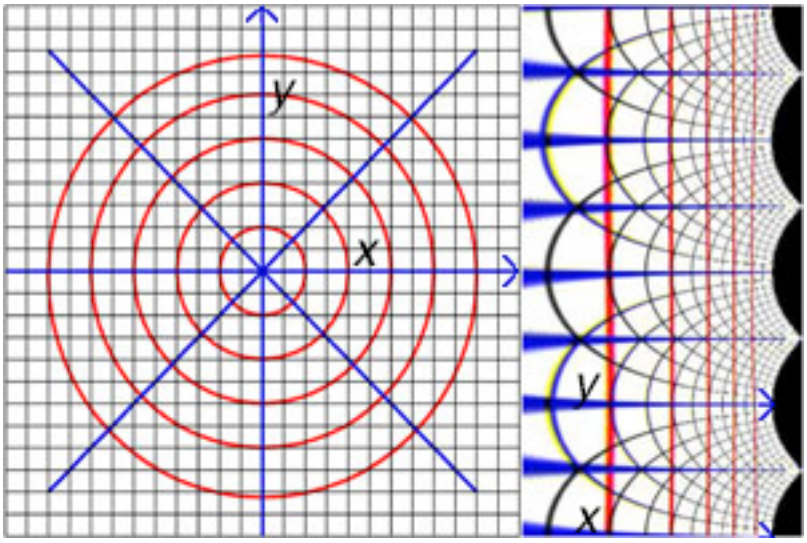

FIGURE 3.5: *Transformation log polaire centrée sur l'image de gauche. La transformée à droite représente le centre de la rétine à gauche et ses extrémités à droite [6].*

Mathématiquement, les coordonnées log-polaires sont représentées par la paire (ρ, θ) où ρ est le logarithme de la distance entre le centre de transformation et le point et θ est l'angle entre la ligne de référence et le point. Les coordonnées cartésiennes se transforment en coordonnées log-polaires selon la formule suivante :

$$\begin{aligned} \rho &= log\sqrt{x^2 + y^2} \\ \theta &= arctan\frac{y}{x} \text{ pour } x \in \mathbf{R}_{\neq 0} \end{aligned} \tag{3.1}$$

Et réciproquement, la transformation des coordonnées log-polaires aux coordonnées

cartésiennes se fait par la formule suivante :

$$x = e^{\rho} cos\theta$$
$$y = e^{\rho} sin\theta \tag{3.2}$$

3.3.1 Les mouvements saccadiques, déplacement du centres d'intérêt sur l'image

Lorsque l'intérêt de la personne se porte sur un autre point de la scène mais qui est hors de la fovéa, cette structure de la rétine incite l'oeil à se centrer sur ce point d'intérêt, et ainsi effectuer un déplacement, pour en obtenir plus d'informations. Ce phénomène s'appelle la fovéation. On parle de mouvement saccadique de l'œil, pour décrire ces mouvements lorsqu'ils sont rapides et de l'ordre d'une fraction de degré sur l'arc visuel de l'oeil. Les saccades de l'oeil sont nécessaires à une vision normale, en effet, si une image est projetée sur la rétine au même endroit, celle-ci disparaîtra progressivement et rapidement de la vision [13], ceci ayant trait à l'adaptabilité de l'œil face au stimulus lumineux. Ces mouvements de l'oeil sont observables (cf. Fig. 3.6), on note des mouvements plus courts et rapides dans les zones plus complexes de l'image et le contraire dans les zones plus homogènes.

---

3. Crédits : Yarbus, A. L. (1967) Eye Movements and Vision. Basil Haigh (trans.). New York : Plenum Press.

FIGURE 3.6: *L'image à gauche est présentée pendant* 2 *minutes à une personne dont les mouvements de l'œil sont retranscrit sur le diagramme à gauche*[3].

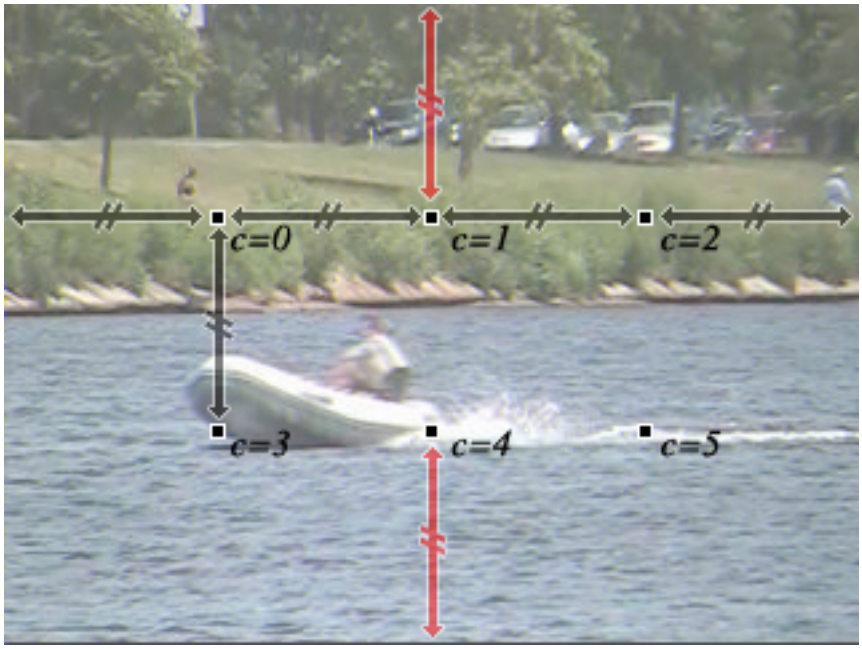

FIGURE 3.7: *Répartition des* 6 *centres des transformations log-polaires.* $C = 6$ *et* $c = \{0, ..., C-1\}$

Une scène en mouvement est donc recomposée par le cerveau à travers la fusion de plusieurs vues de la scène, centrée sur différents points d'intérêts. Pour le SVH, l'analyse d'un objet en mouvement se fait à travers plusieurs densités différentes d'échantillonnage spatial. Pour modéliser le mouvement de l'oeil, on propose donc de répartir sur l'image plusieurs centres à partir desquels seront individuellement issues une transformation log-polaire. Ces centres seront répartis sur un repère cartésien orthonormé de dimension $3 * 2$, centré sur l'image, tel que décrit en Fig. 3.7. On simule donc six déplacements de l'oeil pour chaque nouvelle image entrante avant de procéder à la détection de mouvements (cf. Fig. 3.8.).

Après avoir fait le traitement et la détection de mouvements sur ces séquences issues des transformées, il restera à appliquer la transformée inverse pour passer de la vue corticale à la vue originale.

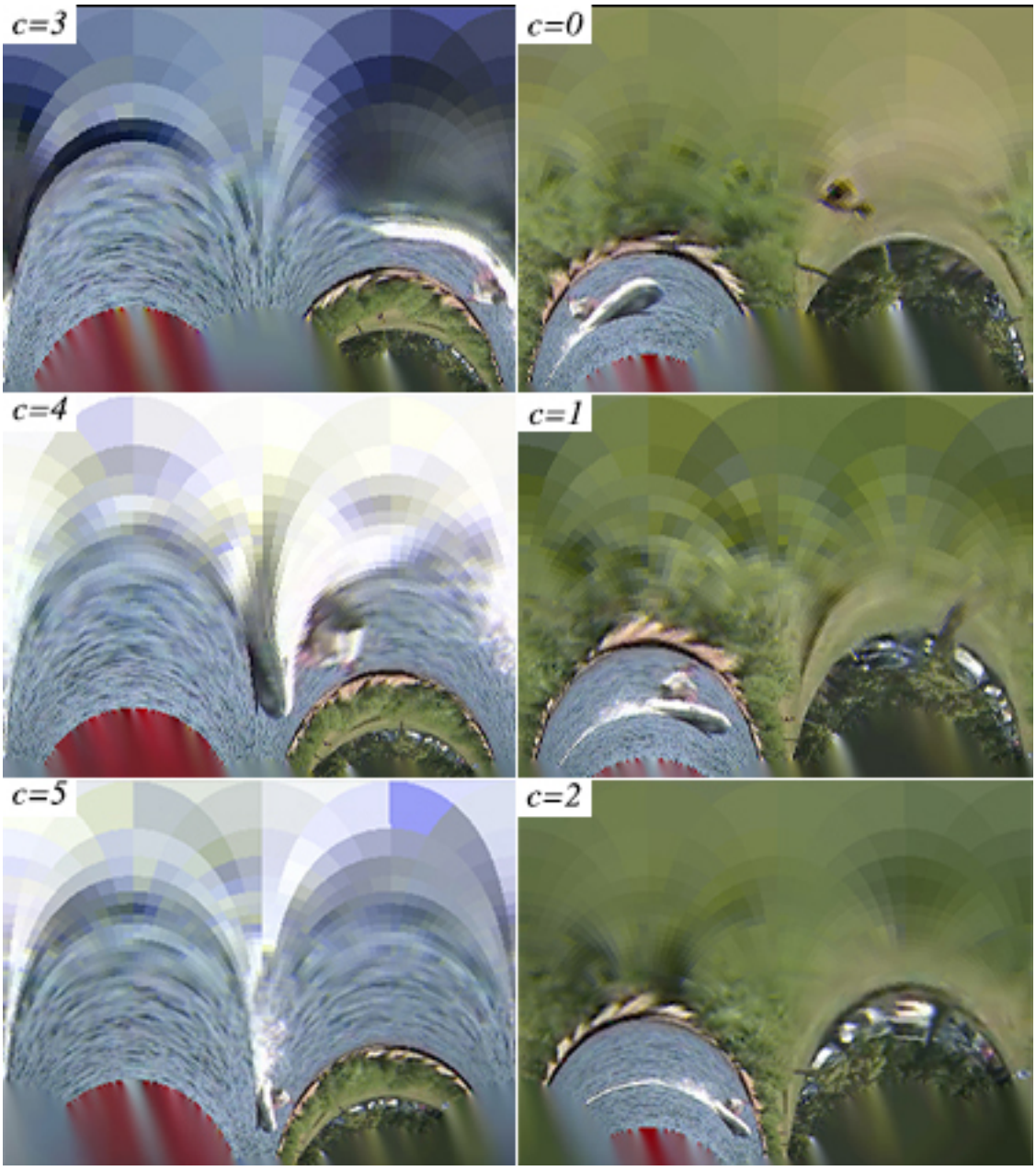

FIGURE 3.8: *Transformations log polaire issues des 6 centres répartis sur l'image (cf. Fig. 3.7). Sur chaque imagette, les données proches du centre sont affichées vers le haut et les plus éloignées du centre, en bas.*

3.4 Filtrage spatial et transformation inverse

La transformation de l'image génère 6 nouvelles séquences qui seront ensuite traitées par l'algorithme de détection de mouvement de Z. Zivkovic. Ainsi les séquences résultantes seront dans un format binaire (cf. Fig. 3.9). Dans notre cas, la transformation log-polaire a concrètement pour principal intérêt l'agrégation de détails à leur environnement, ceci permettant de limiter les erreurs de détection sur les objets en mouvement, ou éviter la détection d'objets non significatifs qui composent l'arrière-plan dynamique (comme une branche en mouvement ou le mouvement non significatif des vagues).

Cet effet inhérent à la transformation log-polaire peut être renforcé par du filtrage spatial (cf. Fig 3.9-3.13.). Puisque l'on travaille sur l'image corticale, ces filtrages auront un effet croissant vers l'extérieure des centres de transformation, ce qui sera notamment visible sur l'image normale une fois la transformation inverse appliquée (cf. Fig. 3.14). On peut noter les faux positifs déclenchés par les vagues à la surface de l'eau et les faux négatifs dans la silhouette du bateau (cf. Fig 3.13). Dépendamment de la position de l'oeil virtuel, les faux positifs et négatifs se réduisent à la taille d'un pixel lorsqu'ils sont en bordure de rétine. Dans notre application, un filtre spatial médian (cf. Fig. 3.13) sur ces transformées log-polaires permet donc de supprimer, dans les zones densément peuplées d'une même classe (fixe ou mobile), les plus fines irrégularités. De même, on peut chercher à exagérer l'agrégation des objets relativement proches en périphérie, et/ou éventuellement effacer des faux négatifs d'une zone représentant la silhouette d'un objet mobile. Pour cela on procède à une inflation de la zone en mouvement (cf Fig. 3.11) puis à une inflation de la zone immobile (cf. Fig. 3.12) par l'utilisation d'un filtre morphologique.

Après la transformation log-polaire inverse (cf. Fig. 3.12), on note par rapport à l'utilisation de GMM (cf. Fig. 3.9(a)) la disparition presque totale des faux négatifs sur la silhouette du bateau, et vers l'extérieure de chaque centre de transformation,

les faux positifs disparaissent. Cependant à la périphérie des centres, des détails se perdent dans les silhouettes détectées à cause de la diminution de la résolution spatiale.

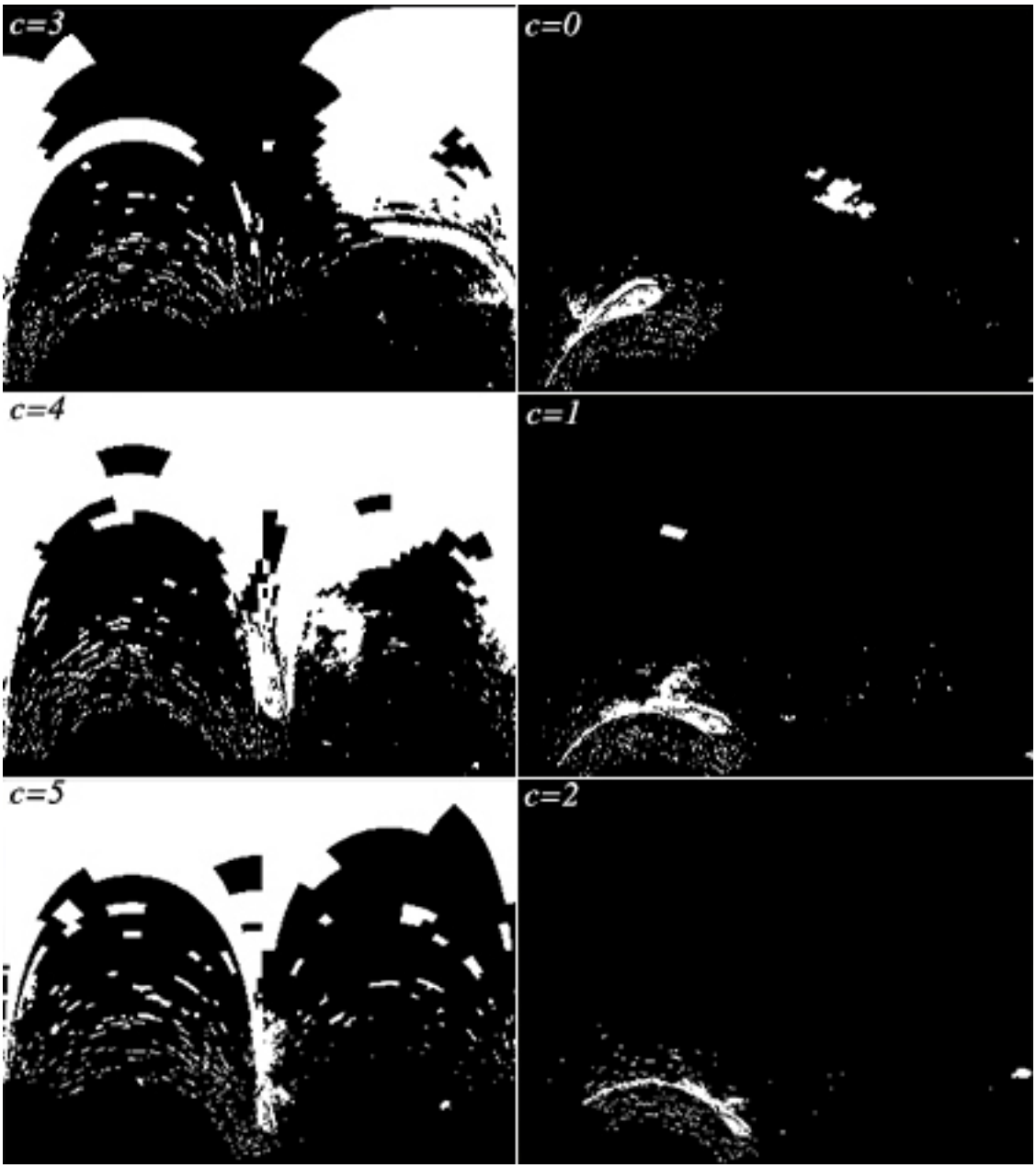

FIGURE 3.9: *Résultats de l'algorithme GMM de Z.Zivkovic appliqué individuellement sur chaque séquence.*

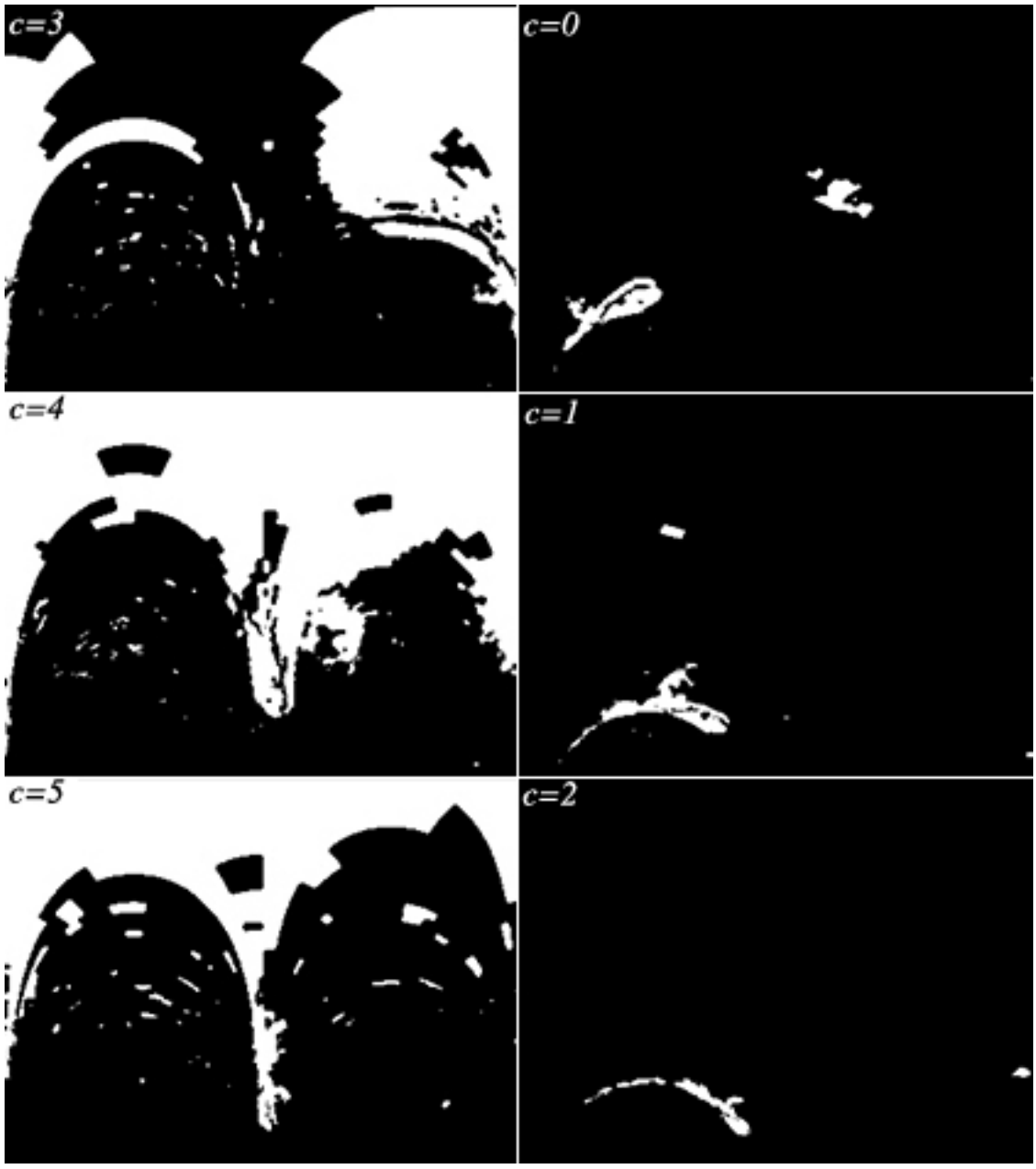

FIGURE 3.10: *Résultat du filtrage médian sur un voisinage de 3*3 pixels.*

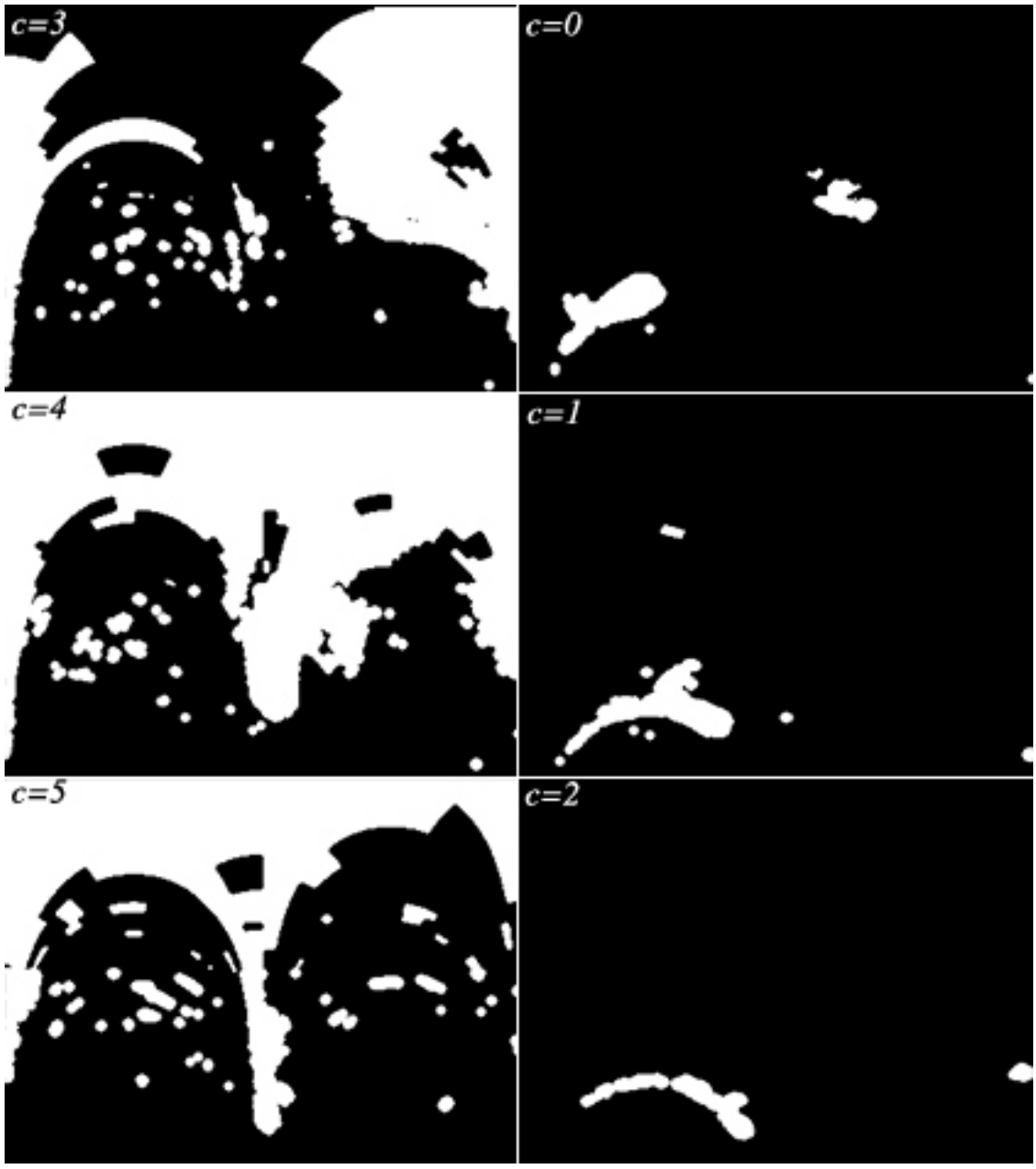

FIGURE 3.11: *Résultat de l'inflation de la zone mobile par un filtre d'un rayon de 1.5 pixels.*

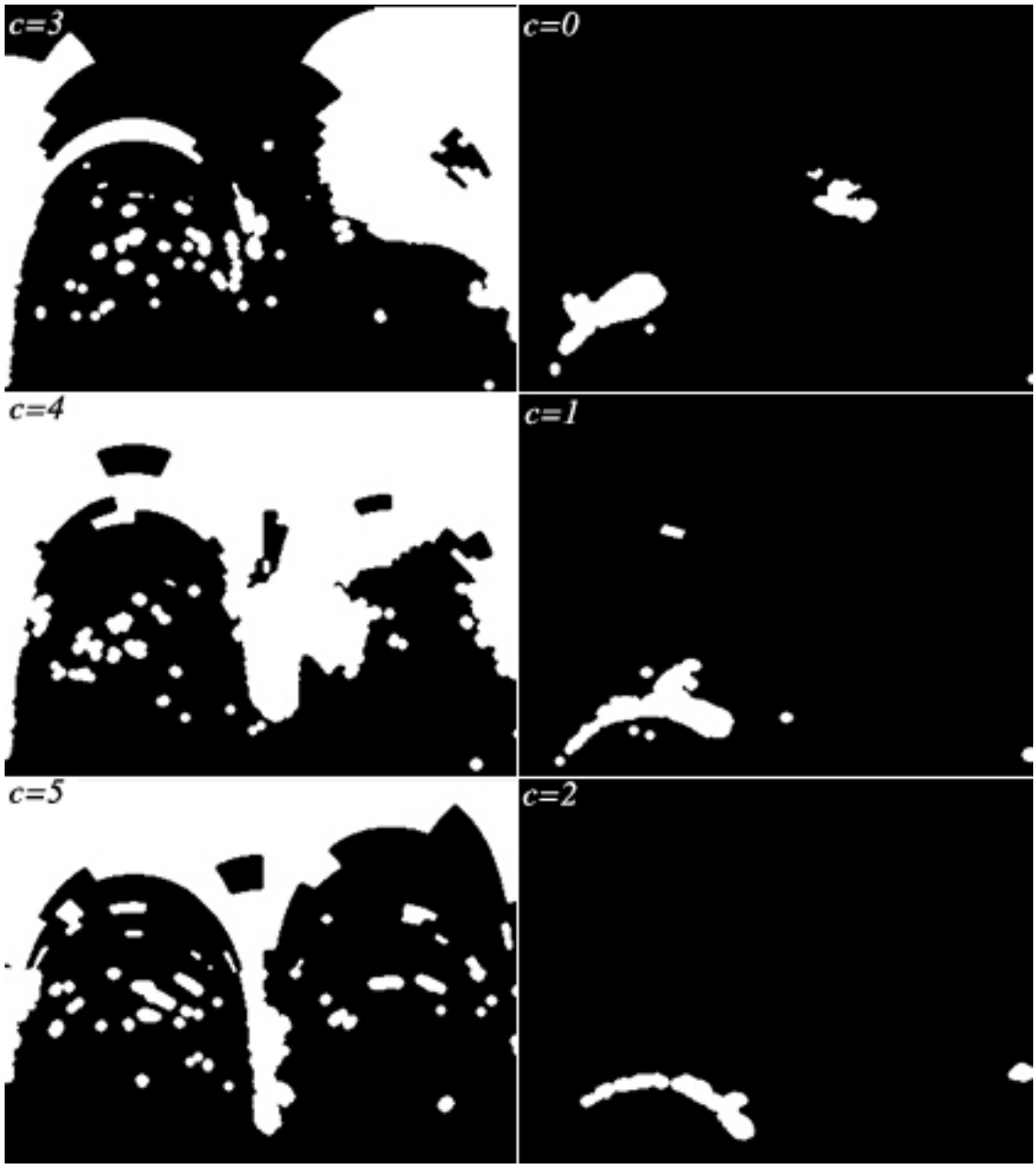

FIGURE 3.12: *Dépendamment de leur forme, certaines zones en faux négatif dans des objets en mouvement ne seront pas effacées par les filtres précédents. Cet algorithme fusionne les régions noires plus petites qu'une surface de 15 pixels avec les zones blanches environnantes. Il s'agit de les effacer avant de contracter les silhouettes pour qu'elle retrouvent leur taille originale.*

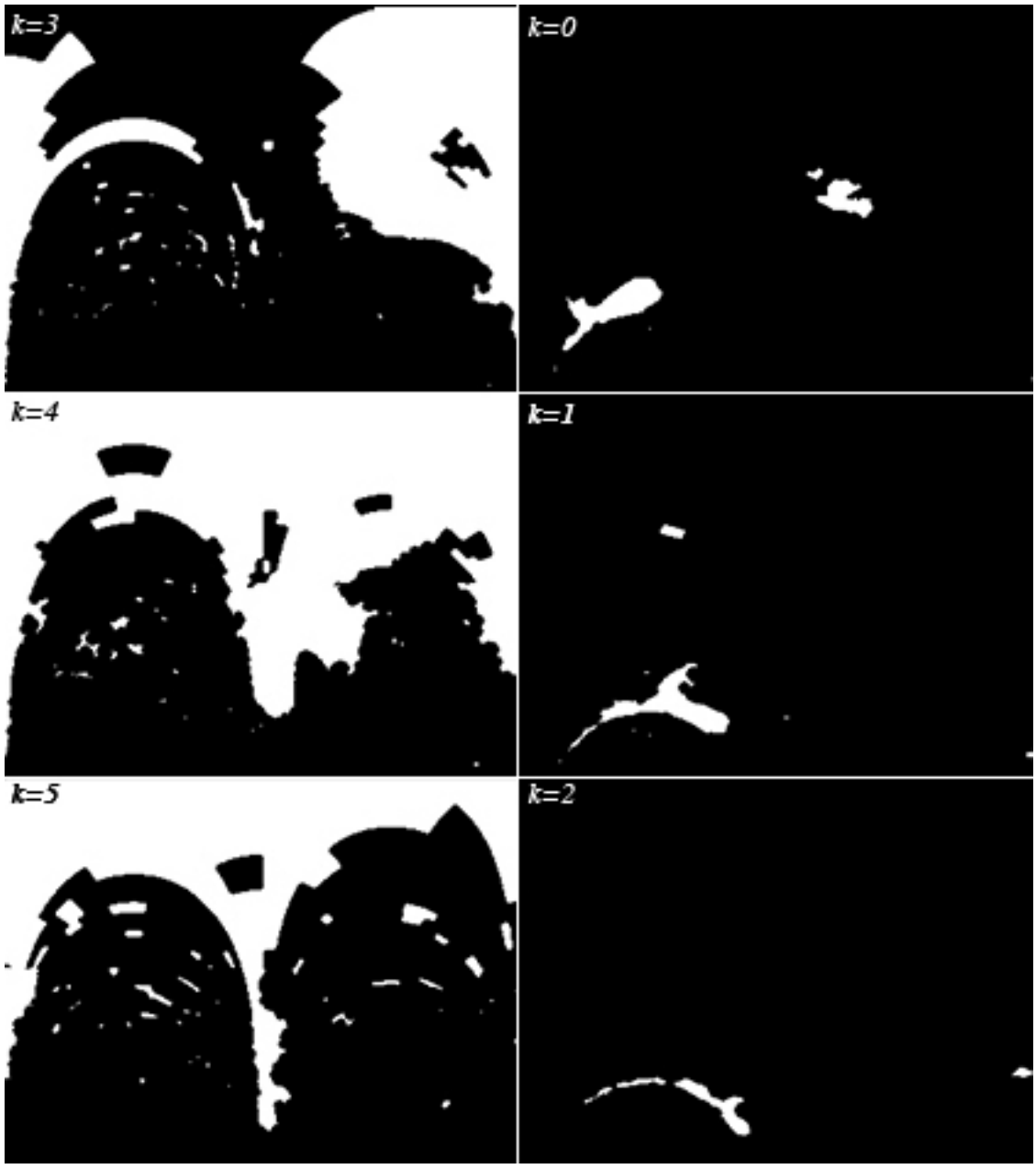

FIGURE 3.13: *Résultat du retour à la taille normale après inflation de la zone noire avec un filtre de 1.5 pixels de rayon.*

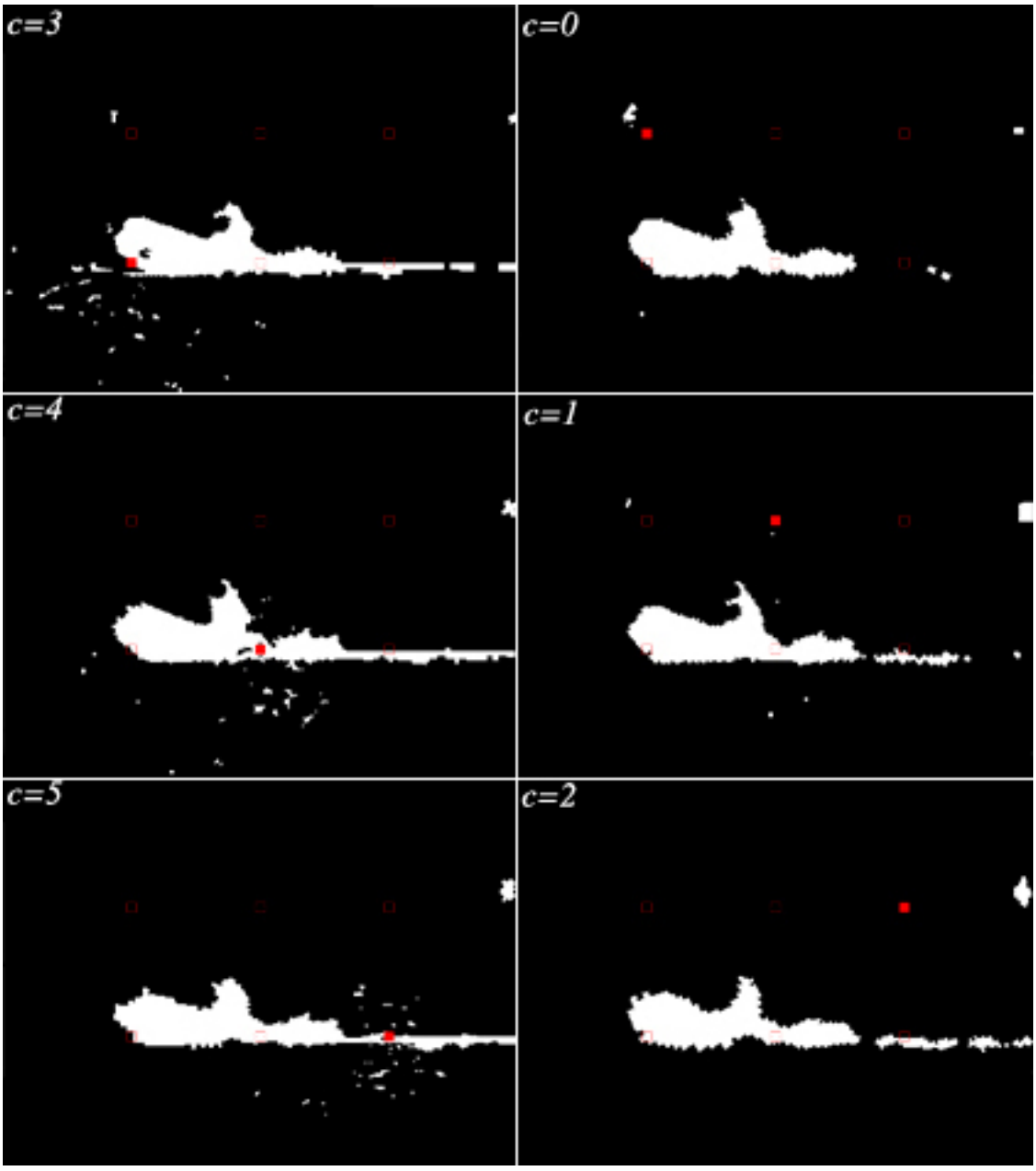

FIGURE 3.14: *Résultats des transformations log-polaire inverse en chaque centre. Le centre de la transformation est marqué en rouge sur chaque image pour faciliter la lecture.*

3.5 Conclusion

Le prétraitement par transformée log-polaire des images de la séquence et les procédés de filtrages mis en place des images segmentées obtenues permettent d'éviter certains défauts de l'algorithme GMM de Z.Zivkovic comme les faux positifs et négatifs, tout en conservant une précision nécessaire à la détection du mouvement. Néanmoins, le résultat obtenu après filtrage et transformation inverse d'une seule transformation log-polaire implique qu'une étape de fusion des résultats de segmentation (pour les différents points de vue ou transformations log-polaires) devra être réalisée, *in fine*, pour obtenir une carte de détection de mouvement fiable et précise. Cette dernière étape de fusion est biologiquement cohérente avec l'étape d'intégration (trans-)saccadique du SVH permettant d'intégrer ou de combiner les différentes interprétations visuelles obtenues selon les différents points de vue (ou transformations log-polaires).

Chapitre 4

REVUE DES MÉTHODES DE VISION UTILISANT LA TRANSFORMÉE LOG-POLAIRE

4.1 Introduction

La représentation en coordonnées log-polaires d'une image est étudiée depuis plus de quarante ans et est exploitée dans différents algorithmes de traitement d'images principalement pour sa ressemblance avec la disposition des capteurs photosensibles de la rétine. Différentes propriétés de cette représentation seront mises en avant dans ce chapitre à travers quelques exemples qui illustreront l'état de l'art et son utilisation dans quelques applications possibles en traitement d'images et vision par ordinateur.

4.2 Évaluer une rotation ou un redimensionnement

Un système de coordonnées log-polaires comprend un point d'origine et un axe polaire. Chaque point de ce système est défini par deux réels : le logarithme de la distance (usuellement : ρ) entre l'origine et ce point et la mesure de l'angle (usuellement : θ) entre l'axe polaire et la droite qui passe par l'origine et par ce point . Le système de coordonnées logs-polaires est par nature $2\,\pi$ périodique en suivant les coordonnées angulaires. Ainsi une rotation de l'image originale sera interprétée comme une simple translation sur l'image log-polaire, suivant l'axe des coordonnées radiales θ. De façon similaire, augmenter ou réduire les dimensions de l'image originale va compresser ou étirer les données sur l'axe ρ. L'appréhension d'une image à travers sa transformée log-polaire, obtenue par échantillonnage *via* un système centré sur l'image tel que décrit en Fig 3.4 et Fig 3.5, simplifie ainsi la modélisation des redimensionnements et

rotations.

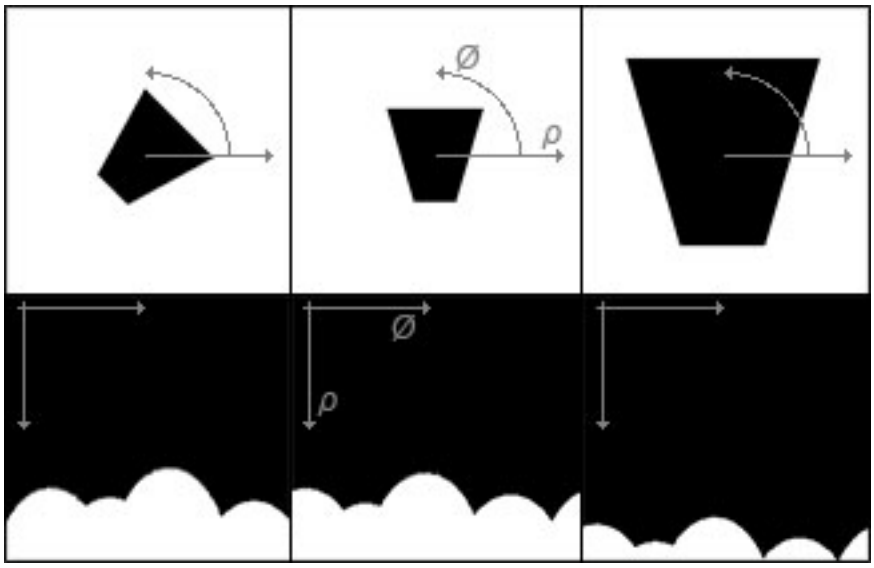

FIGURE 4.1: *Effets de la rotation et du redimensionnement d'une image sur la transformée log-polaire en son centre.*

Ainsi la rotation et le redimensionnement qui transforment les données sur les deux axes dans un système de coordonnées cartésiennes agissent sur un seul dans le système de coordonnés logs-polaires (cf. Fig. 4.1). Cette propriété est quelquefois utilisée pour rendre les techniques de classification d'images (ou d'objets extraits dans celle-ci) invariantes (ou plus robustes) aux changements d'échelles et/ou de rotations.

4.2.1 Classification et recalage robustes d'images

Dans [14] l'utilisation de la transformée log-polaire sur des image peu complexes (images simples comme celles représentant différents visages) permet de proposer une technique de classification qui a la propriété d'être invariante à toute combinaison de rotation et de changement d'échelle. Une technique similaire a également été proposé dans [15] permettant une classification robuste des textures (par comparaison à une signature propre à chaque texture) qui est invariante à une rotation ou un re-

dimensionnement. [14,15] utilisent les caractéristiques de décalage de la transformée log-polaire pour proposer une technique de recalage d'images permettant d'estimer, par mesure de corrélation, la transformation géométrique existante entre chaque pair d'images, provenant de différents repères.

4.2.2 Watermarking ou filigrane

Le watermarking (tatouage ou filigrane numérique) est une technique permettant la superposition de données (de copyright ou d'autres messages de vérification) à un document numérique original de façon à ce qu'elles puissent être récupérées par la suite. On distingue généralement les tatouages invisibles ou visibles (les agences de photos ajoutent, à leurs images, un tatouage visible afin d'éviter que ces versions de prévisualisation, basse résolution, ne se substituent aux versions hautes résolutions payantes).

Les méthodes détaillées dans [7] (cf. Fig 4.2), et similairement dans [16,17], proposent, pour les images, un watermarking non perceptible par le SVH et insensible aux rotations et redimensionnements. Pour en éviter une perception par le SVH, l'intégration du watermark se fait sur le spectre de magnitude obtenu par la transformée de Fourier discrète de l'image. Plus précisément, les données à cacher sont intégrées sur la transformée dans le domaine log-polaire du spectre de magnitude centré obtenu. Le watermarking est ainsi rendu insensible aux rotations et redimensionnements dans sa relecture et les données sont retrouvées en comparant simplement le spectre de magnitude de l'image marquée à celui de l'originale.

4.3 Réduction de données

L'article [16] montre que la rétine, tout en gardant un large champ visuel, permet de concentrer l'attention sur une zone centrale dont la résolution est plus importante que sur le reste de l'image, et ainsi restreindre les données à analyser. Cette

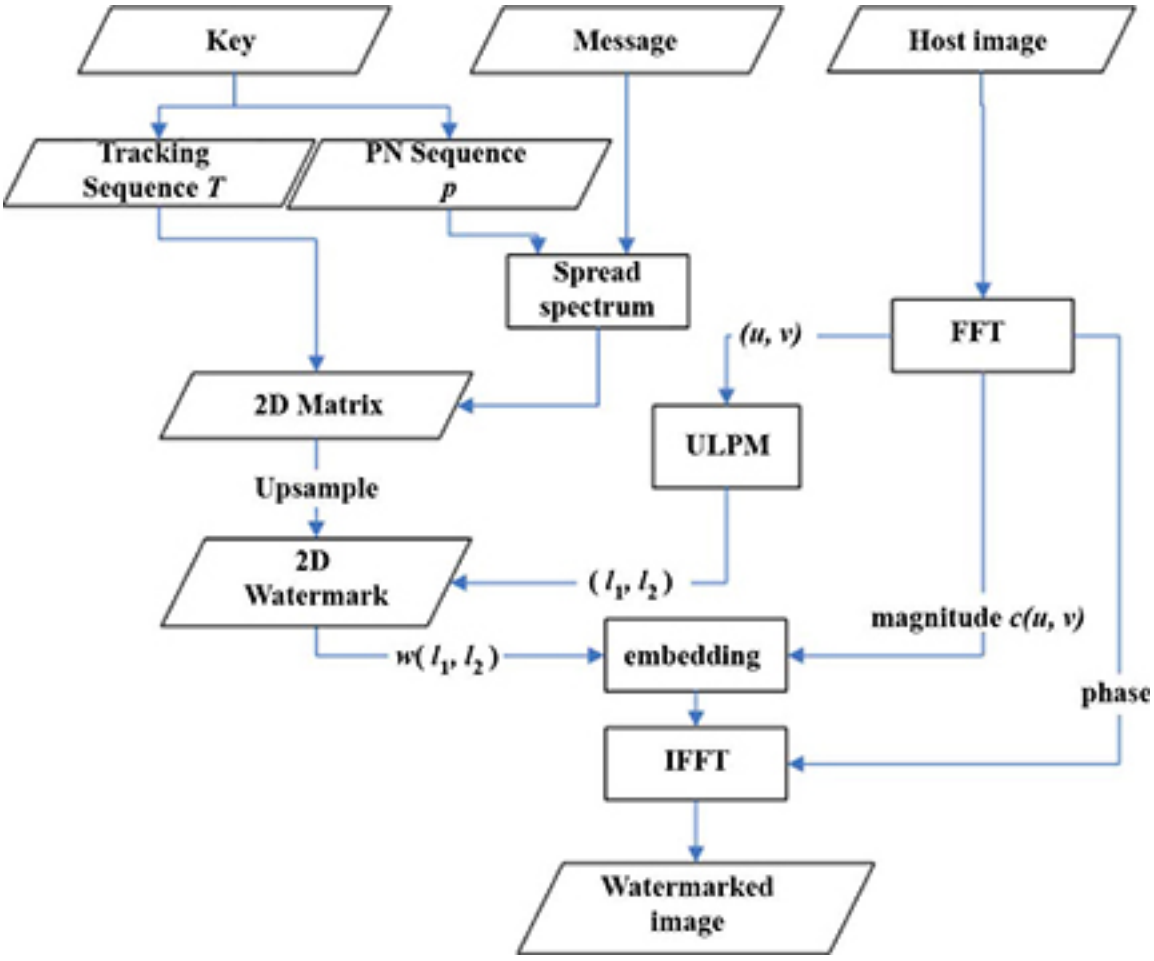

FIGURE 4.2: *Processus de watermarking issu de [7].*

caractéristique est intéressante si on la compare aux prises de vues générées par les caméras classiques qui offrent une précision ou une résolution homogène sur toute la surface de l'image. En effet, la vision par ordinateur implique souvent une contrainte temporelle où le système établit doit réagir en temps réel à une situation présente et, pour réduire efficacement le temps de calcul, s'intéresser prioritairement aux données les plus importantes. C'est cette caractéristique essentielle de la représentation et vision log-polaire que notre algorithme exploitera pour réduire l'importance des détails d'une scène (relativement au centre de l'image) dans le cadre d'une détection des

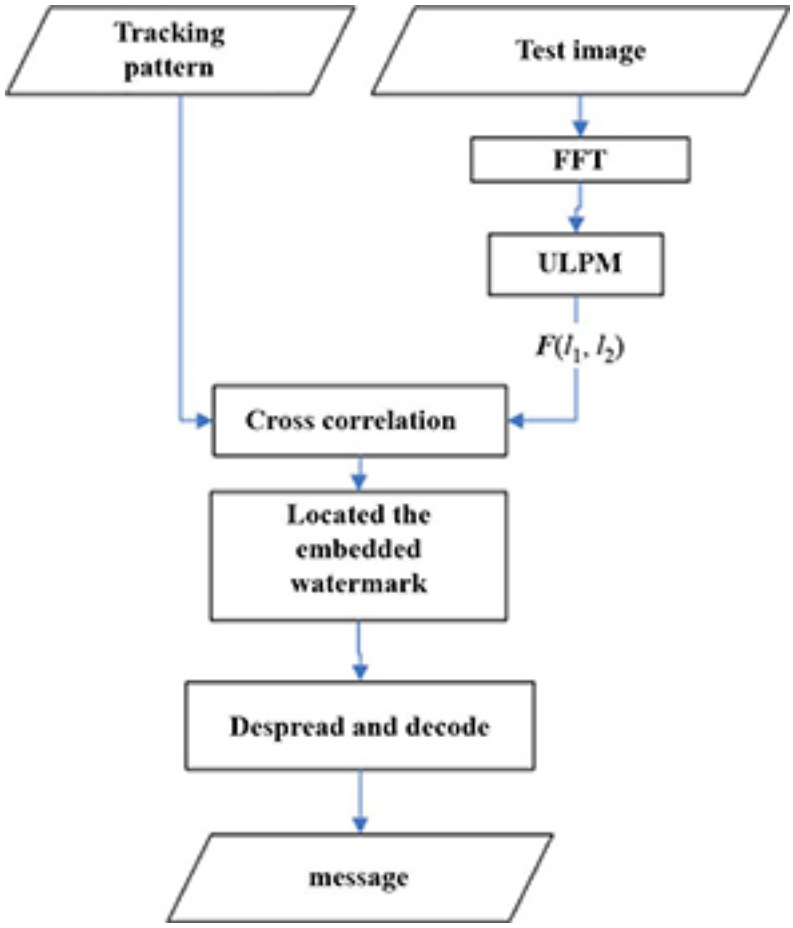

FIGURE 4.3: *Processus d'extraction du watermarking issu de [7]. La transformée log-polaire intervient aussi dans le processus d'extraction.*

zones mobiles dans une séquence d'images.

4.3.1 Vision stéréoscopique : contrôle de la vergence

Un des grands intérêts pour le SVH de la vision stéréoscopique est l'appréciation des distances. Lorsqu'un objet de la scène est ciblé, le phénomène de vergence fait converger la direction des deux yeux pour le maintenir au centre de la vision et en obtenir deux images montrant peu de disparité. Ce phénomène crée ainsi le système triangulaire décrit (cf. Fig. 4.4) duquel est extrait la distance entre les yeux et la cible.

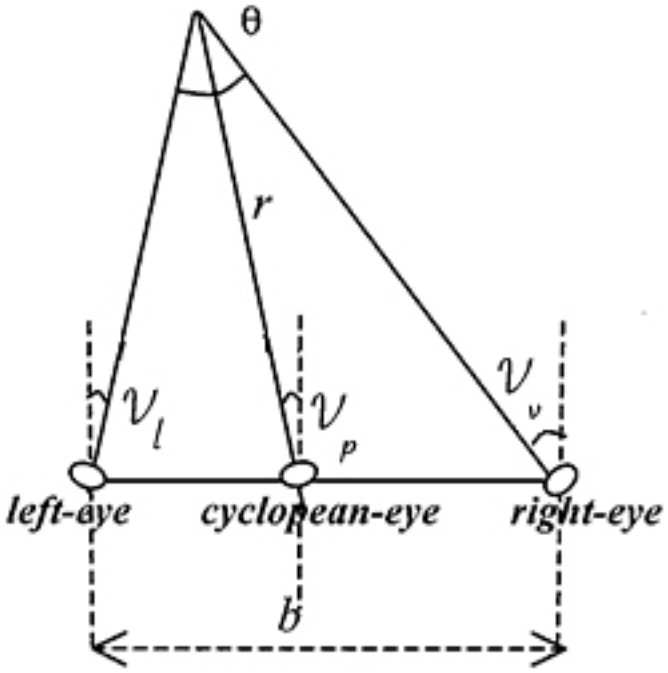

FIGURE 4.4: *Système de vision stéréoscopique [8]. Il y est décrit l'angle de vergence* θ.

Le contrôle de la vergence est le problème traité dans [8]. Le système détaillé se base sur un modèle inspiré biologiquement à travers l'utilisation de caméras dont le capteur est construit sur le modèle topologique rétinien. Ici, le contrôle de la vergence se base sur l'estimation de la disparité entre les deux images, ce qui consiste à déterminer la différence de position entre les pixels de l'image de gauche et de droite correspondants à un même objet dans l'espace. Amener la disparité à un minimum permet donc d'obtenir l'angle de vergence correct pour un objet cible.

Il est expliqué dans [8] que pour réduire le temps de calcul d'une estimation de disparité entre deux images cartésiennes, le calcul se fait sur une cible restreinte dans ces images, soit l'objet d'intérêt. Dans ce cas, la cible doit être cernée au préalable, elle est extraite par segmentation sur les deux images d'entrée, augmentant fortement le temps de calcul. Or, sur la caméra log-polaire, lorsque l'objet cible entre au centre

du capteur il couvre plus de pixels que le reste de l'image et est implicitement ciblé, la disparité par rapport à la vue de l'autre caméra peut ensuite être calculée. Inversement, les pixels à l'extérieure de la rétine simulée ont un impact diminué. L'utilisation d'images log-polaires permet donc d'éviter l'étape de segmentation des images.

4.4 Détection de contours

L'article [6] présente un algorithme de détection de contour basé sur un modèle d'inspiration biologique duquel le sujet de ce mémoire s'inspire. Il y est mis en avant que les (portions de) courbes composant les contours dans une image peuvent toutes être perçues comme des segments (de droite) selon un centre de transformation log-polaire bien choisi, facilitant ainsi leur détection par un algorithme robuste de détection de lignes (ou de segments de droite). La transformée log-polaire est donc réalisée en plusieurs points de l'image (suivant sa diagonale), mimant ainsi les mouvements saccadés de la rétine de l'oeil et offrant une diversité de points de vue de l'image. Les segments ainsi extraits à travers ses différents points de vue sont ensuite retransformées dans le domaine cartésien (sous forme de courbes) puis fusionnés (pondérés et moyennés dans cette application) pour obtenir, *in fine*, une carte de détection de contour fiable des différents objets contenus dans l'image. La validation de cette méthode sur la base de Berkeley a montré tout l'intérêt et l'efficacité de ce modèle de détection de contour.

4.5 Conclusion

Ces différents travaux montrent que les déformations de l'image entraînées par la transformation log-polaire permettent divers traitements qui n'auraient pas été possibles avec le système de coordonnées cartésiennes, tout en maintenant la rapidité de traitement grâce à la compression de données qu'il produit. On peut aussi envisa-

ger au sein d'un même projet une complémentarité entre certaines de ces techniques puisqu'elles se basent toutes sur le même principe. Tel que décrit dans le paragraphe précédent, la multiplication des transformations en différents centres mime le fonctionnement de l'oeil humain, générant ainsi plusieurs images à partir d'une seule. On multiplie les points de vue pour permettre à l'algorithme de détection de mouvement [1] de ne pas y reproduire les mêmes erreurs (cf. Chapitre 3). Le chapitre suivant va permettre de détailler le processus employé pour la fusion des résultats de la détection de mouvement sur ces points de vue.

Chapitre 5

FUSION DE POINTS DE VUE

5.1 Fusion médiane des points de vue

5.1.1 Introduction et Définitions

Comme nous l'avons déjà mentionné, la détection du mouvement, réalisée sur une séquence d'images transformées en coordonnées log-polaires (pour différents centres de cette transformation), permet d'éviter certains défauts de l'algorithme GMM de Z.Zivkovic (cf. Fig 3.14) et s'inspire directement du processus de saccade oculaire du SVH réalisé par l'oeil pour l'exploration visuelle des parties importantes d'une scène combiné au processus de projection de cette image sur le cortex visuel.

On désigne, par la suite, ces images (ou transformations log-polaires pour un point de vue particulier) par le terme *composantes (de fusion)*. On désigne le nombre de composantes total par C et chaque composante par c où $c = \{0, ..., C-1\}$. Chaque composante est associée à la position du point de transformation o (ou point de vue) duquel elle est issue sur l'image originale. Ce point de transformation étant le centre du repère log-polaire appliqué à l'image, et ainsi le centre de la rétine simulée, on s'y réfère directement comme étant le *centre de transformation d'une composante*, qui, on le rappelle, n'est pas nécessairement le centre de l'image.

5.1.2 L'intérêt de la fusion

Les résultats obtenus pour chaque transformation log-polaires ou point de vue différent (cf. Fig 3.14) doivent être maintenant fusionnés pour obtenir une carte de détection de mouvement fiable et précise et imiter ainsi biologiquement, l'étape

d'intégration trans-saccadique du SVH permettant d'intégrer ou de combiner les différentes interprétations visuelles acquises selon les différents points de vue.

Algorithme 1

Fusion médiane des composantes

$\mathrm{R}(x, y)$	Valeur pour un pixel (x, y) de l'image résultante de la fusion
$\mathrm{CP}(x, y, c)$	Valeur pour un pixel (x, y) de la composantes c à fusionner
r	Rayon $r = 1.5$

pour *chaque* $R(x, y)$ **faire**

  pour *Pour chaque pixel* $CP(i, j, c)$ *de chaque composante où* (i, j) *est dans le voisinage* r *de* (x, y) **faire**

    - **Si** ($\mathrm{CP}(i, j, c)$ ==BLANC) $compte = compte + 1$
    - **Sinon** $compte = compte - 1$

  - **Si** ($compte < 0$) $\mathrm{R}(x, y)$ =NOIR
  - **Sinon** $\mathrm{R}(x, y)$ =BLANC

Algorithme 1: Fusion médiane des différentes composantes pour l'estimation de la carte finale de détection de mouvement R (dans laquelle les pixels BLANCS représentent les zones mobiles et les pixels NOIRS les zones immobiles)

Il est important de rappeler que les composantes de fusion peuvent se définir comme des résultats de détection de mouvement présentant une fovéation. Ceci signifie que sur une composante, la détection du mouvement présente une résolution qui décroît à mesure que l'on s'écarte de son centre de transformation. En se reposant sur ce principe, le chapitre 3 a montré que les erreurs de détection, inhérentes

à l'algorithme de détection de mouvement utilisé (GMM), peuvent être restreintes à une zone entourant le centre de transformation des composantes (cf. Fig 3.14 dans laquelle on peut remarquer un bruit dans la détection qui est présent sur chaque composante autour de son centre de transformation). On va donc chercher à réduire ces erreurs dans le processus de fusion (sachant que si on superpose toutes ses composantes, ces zones erronées ne se recouvrent pas ou peu entre elles). Cette fusion des différentes composantes devra combiner des données (de détection de mouvement) détectées à différents niveaux de résolution (*i.e.*, obtenues à différentes distances du point de transformation), pour obtenir une carte finale de détection de mouvement fiable et précise et imiter ainsi le processus d'intégration trans-saccadique du SVH.

Une première approche est donc celle de la fusion médiane entre toutes les composantes, associée à un filtre spatial, tel que définie dans l'algorithme 1.

5.2 Fusion avec pondération spatiale des composantes

5.2.1 Limites de la fusion médiane

La fusion médiane va permettre de filtrer le bruit présent sur les composantes (dans les zones citées précédemment) dans le résultat final de détection. Cependant, lorsqu'on s'écarte du centre de transformation d'une composante, et toujours à cause du principe de la fovéation, si la détection présente moins de bruit, elle va aussi graduellement perdre en détail (cf. Fig 5.1). Les objets mobiles prennent une silhouette moins précise, ce qui va contribuer à générer des faux positifs ou des faux négatifs sur le périmètre des objets détectés.

5.2.2 Apports de la pondération des composantes

Rappelons que chaque centre de transformation est censé représenter en fait une partie importante de la scène qui est explorée (lors du processus de saccade oculaire)

et que la transformée log-polaire associé à ce centre permettra une détection des étiquettes de mouvement (mobile/immobile), dans cette partie de la scène ou de l'image, avec un niveau de résolution et donc de détection plus importante. Lors de la fusion, il est donc important d'attribuer aux composantes à fusionner un poids d'autant plus élevé que celle-ci sera associée à un centre de transformation qui sera proche d'un pixel du résultat de détection final.

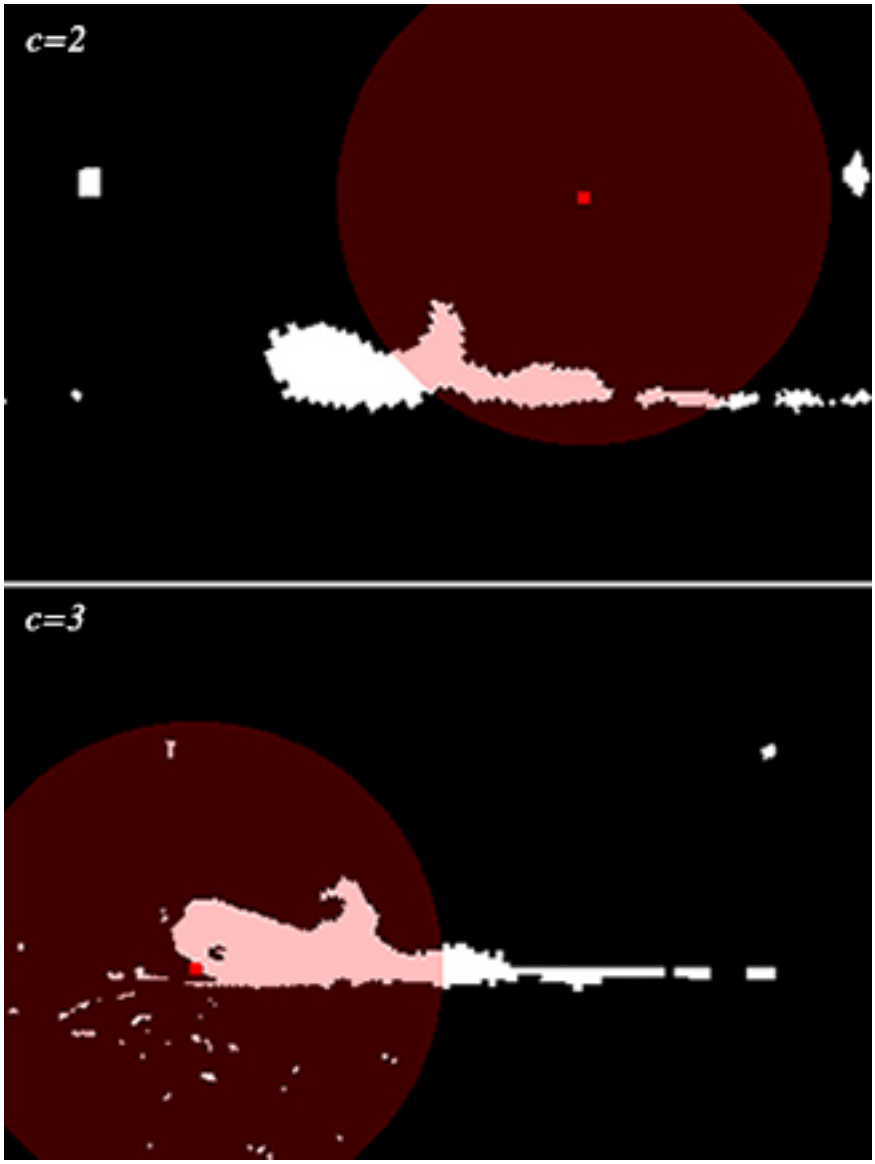

FIGURE 5.1: *Extrait de la Fig. 3.14 avec le marquage en rouge des zones entourant le centre de transformation des composantes, qui présentent du bruit mais une détection des contours plus précise.*

Le filtrage médian permettra toujours de filtrer le bruit de détection des différentes composantes mais cette pondération permettra aussi de prendre en compte les différents niveaux de résolutions pour lesquels les étiquettes de mouvement (mobile/immobile)

ont été détectés par l'algorithme. Autrement dit, le principe de fusion pondérée va favoriser les composantes dont les centres de transformation sont dans un voisinage du pixel fusionné.

5.2.3 Fonction de pondération

Les pixels de coordonnées x et y des composantes $c = \{0, ..., C-1\}$ seront individuellement pondérés dans la fusion par une gaussienne $W_{c,\beta}(x, y)$. W est fonction de la distance $D_c(x, y)$ entre le point (x, y) et le centre de la transformation associé o_c ainsi que de la distance minimale D_{min} entre deux centres, [1] et d'un coefficient de fusion $\beta(> 0)$. Pour une composante c, on a :

$$\begin{aligned} a_\beta &= \frac{1}{D_{min} \times \beta} \\ D_c(x,y) &= \sqrt{\left(x_{o_c} - x\right)^2 + \left(y_{o_c} - y\right)^2} \\ W_{c,\beta}(x,y) &= \left(D_c(x,y) \times a_\beta + 1\right) \times \frac{1}{\exp(D_c(x,y) \times a_\beta)} \end{aligned} \tag{5.1}$$

Plus le coefficient β s'éloigne de 0, plus les pixels d'une composante éloignée de son centre de transformation auront un poids important. Augmenter c va donc uniformiser les poids des différentes composantes utilisées dans le processus de fusion (cf. Fig. 5.2, Tab. 5.1).

1. (Cf. Fig.3.7) : $D_{\min} = \frac{\text{largeur de l'image}}{\text{nombre de centres à l'horizontale}+1}$

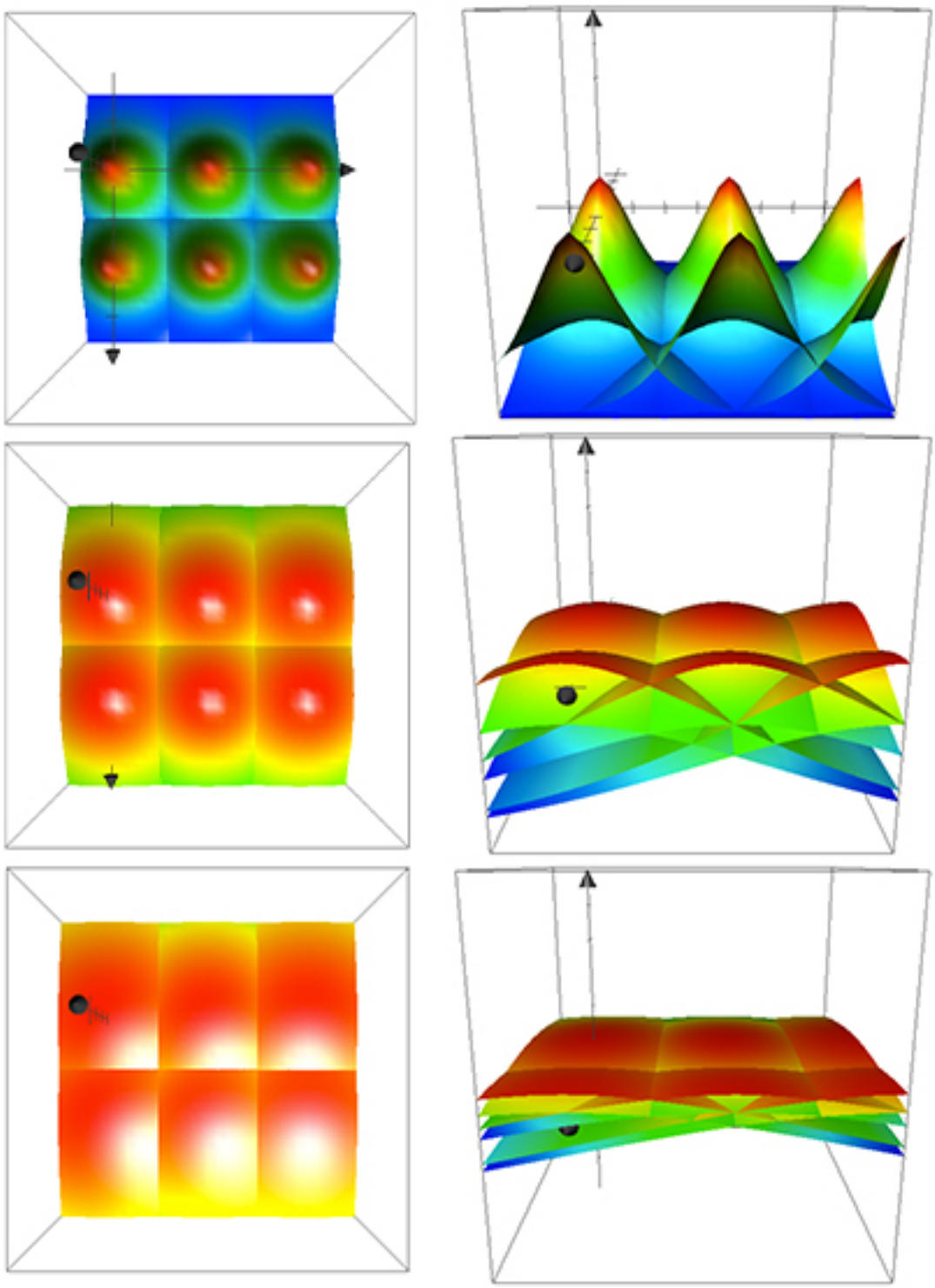

FIGURE 5.2: *Courbe illustrant les poids $W_{c,\beta}(x, y)$ des composantes de la fusion en tout point de l'image selon sa position (x, y) et selon le coefficient β. De haut en bas, $\beta = 0.2$, $\beta = 0.9$, $\beta = 2.0$.*

Pour la fusion en chaque pixel, un poids est attribué à chaque composante. Celui-ci est soustrait ou ajouté à un compteur selon que le pixel donné sur cette composante est respectivement immobile ou mobile. Si ce compteur est positif, alors le pixel résultant de la fusion sera classé mobile, donc avec une étiquette blanche ou au contraire classé immobile (étiquette noire).

Algorithme 2

Fusion pondérée des composantes

$\mathrm{R}(x, y)$	Valeur pour un pixel (x, y) de l'image résultante de la fusion
$\mathrm{CP}(x, y, c)$	Valeur pour un pixel (x, y) de la composantes c à fusionner
$W_{c,\beta}(x, y)$	Fonction de calcul du poids avec le coefficient de fusion β
r	Rayon $r = 1.5$

pour *chaque* $R(x, y)$ **faire**

- poids $= 0$
- **pour** *Pour chaque pixel* $CP(i, j, c)$ *de chaque composante où* (i, j) *est dans le voisinage* r *de* (x, y) **faire**
  - **Si** $(\mathrm{CP}(i, j, c) ==$ BLANC$)$ $\quad poids = poids + W_{c,\beta}(i, j)$
  - **Sinon** $\quad poids = poids - W_{c,\beta}(i, j)$
- **Si** $(poids < 0)$ $R(x, y) =$ NOIR
- **Sinon** $R(x, y) =$ BLANC

Algorithme 2: Fusion pondérée des composantes (cf. Fig. 3.14) pour l'estimation de la carte finale de détection de mouvement R (dans laquelle les pixels BLANCS représentent les zones mobiles et les pixels NOIRS les zones immobiles)

5.3 Restrictions sur le coefficient de fusion

Le tableau suivant permet d'illustrer pour chaque composante, la proportions $W'_{c,\beta}$ qu'elle va avoir dans le cas d'une fusion en un pixel situé sur le centre de la transformation de la composante $k = 3$ (centre bas gauche). Chaque composante reçoit son poids duquel est déduit le pourcentage qu'elle occupe dans la fusion. On peut ainsi se rendre compte du potentiel qu'ont les composantes de centres éloignées dans la modification du résultat de la fusion :

$\beta = 0.20$:

$W_{0,\beta} = 0.040 \quad W'_{0,\beta} = 4\%$	$W_{1,\beta} = 0.007 \quad W'_{1,\beta} = 1\%$	$W_{2,\beta} = 0.000 \quad W'_{2,\beta} = 0\%$
$W_{3,\beta} = 1.000 \quad W'_{3,\beta} =$**92%**	$W_{4,\beta} = 0.040 \quad W'_{4,\beta} = 4\%$	$W_{5,\beta} = 0.000 \quad W'_{5,\beta} = 0\%$

$\beta = 0.45$:

$W_{0,\beta} = 0.349 \quad W'_{0,\beta} = 18\%$	$W_{1,\beta} = 0.179 \quad W'_{1,\beta} = 9\%$	$W_{2,\beta} = 0.041 \quad W'_{2,\beta} = 2\%$
$W_{3,\beta} = 1.000 \quad W'_{3,\beta} =$**50%**	$W_{4,\beta} = 349 \quad W'_{4,\beta} = 18\%$	$W_{5,\beta} = 0.064 \quad W'_{5,\beta} = 3\%$

$\beta = 0.90$:

$W_{0,\beta} = 0.695 \quad W'_{0,\beta} = 20\%$	$W_{1,\beta} = 0.534 \quad W'_{1,\beta} = 15\%$	$W_{2,\beta} = 0.290 \quad W'_{2,\beta} = 8\%$
$W_{3,\beta} = 1.000 \quad W'_{3,\beta} =$**28%**	$W_{4,\beta} = 0.695 \quad W'_{4,\beta} = 20\%$	$W_{5,\beta} = 0.349 \quad W'_{5,\beta} = 10\%$

$\beta = 2.00$:

$W_{0,\beta} = 0.910 \quad W'_{0,\beta} = 18\%$	$W_{1,\beta} = 0.842 \quad W'_{1,\beta} = 17\%$	$W_{2,\beta} = 0.692 \quad W'_{2,\beta} = 14\%$
$W_{3,\beta} = 1.000 \quad W'_{3,\beta} =$**20%**	$W_{4,\beta} = 0.910 \quad W'_{4,\beta} = 18\%$	$W_{5,\beta} = 0.736 \quad W'_{5,\beta} = 14\%$

$\beta \to \infty$ ou fusion médiane :

$W_{0,\beta} \to 1.000 \quad W'_{0,\beta} = 17\%$	$W_{1,\beta} \to 1.000 \quad W'_{1,\beta} = 17\%$	$W_{2,\beta} \to 1.000 \quad W'_{2,\beta} = 17\%$
$W_{3,\beta} \to 1.000 \quad W'_{3,\beta} =$**17%**	$W_{4,\beta} \to 1.000 \quad W'_{4,\beta} = 17\%$	$W_{5,\beta} \to 1.000 \quad W'_{5,\beta} = 17\%$

TABLE 5.1: *$W_{c,\beta}(i,j)$ pour $x = x_{o_3}$ et $y = y_{o_3}$ pour 6 composantes telles que sur Fig. 3.14. Pour une composante c, sa proportion $W'_{c,\beta}$ dans la fusion donnée en pourcentage est calculée par $W'_{c,\beta} = W_{c,\beta}/(\sum_{c'=0}^{C}(W_{c',\beta}))$, où C est le nombre de composantes.*

5.3.1 Minoration du coefficient

Notons que lors du processus de fusion pondérée des composantes, si l'une des composantes à un poids associé dépassant 50% de la somme totale, alors cette composante imposera sa décision sur toutes les autres composantes et elle sera donc la seule prise en compte. Or, l'intérêt de la fusion consiste à considérer au minimum deux composantes pour produire un résultat. Les calculs présentés dans le tableau 5.1 nous indiquent qu'un coefficient trop bas empêche de remplir cette condition essentielle. Cette condition sera respectée, sur l'ensemble de l'image, si le point le plus éloigné des centres de transformation (typiquement $x = 0$ $y = 0$ pour l'organisation décrite en Fig. 3.7.) est pondéré à 50% ou moins sur la composante du centre le plus proche, soit $c = 0$ (puisqu'elle sera nécessairement la plus importante), tel que pour C composantes :

$$W_{0,\beta}(0,0) \;\leq\; \sum_{c=0}^{C} W_{c,\beta}(0,0) \tag{5.2}$$

La qualité de la détection de base influe aussi sur le coefficient minimum à choisir. La précision des composantes dans la zone entourant leur centre de transformation n'étant pas pertinente dans le cas d'une image contenant beaucoup de bruit (cf. Fig 5.1), il faut donner plus de poids dans la fusion aux autres composantes en augmentant le coefficient (et inversement).

5.3.2 Vers un coefficient maximum

Par rapport à une simple fusion médiane on cherche aussi à modérer l'impact des composantes les plus éloignées faces aux plus proches. Cependant augmenter le coefficient de fusion permet d'atténuer la pondération des composantes. Un coefficient trop haut fera tendre le résultat vers celui d'une fusion médiane.

5.3.3 Influence sur le coefficient de la distribution des centres de transformation

Du fait que le calcul des poids est ajusté par rapport à l'écart horizontal entre les centres de transformation (D_{min}), ce calcul prendra indirectement en compte la taille de l'image et celle-ci n'influera pas trop sur cette pondération pour le choix d'un coefficient c. En revanche, une qualité d'image moindre va requérir d'utiliser moins de composantes, l'oeil virtuel ayant à saisir moins de détails mais plutôt une scène dans son ensemble. Moins il y aura de composantes, moins la fusion sera répartie. Pour chaque composante retirée, la proportion maximale d'une composante dans la fusion sera nécessairement augmentée. Pour satisfaire les conditions minorantes et majorantes exprimées précédemment, le choix du coefficient de fusion doit tenir compte de cette organisation des centres de transformation entre eux.

5.4 Résultats de fusion

Les résultats de fusion (cf. Fig. 5.3) mettent en évidence qu'un coefficient trop faible, par exemple $c = 0.20$, ne permet pas de tirer parti de l'intérêt de la fusion sur l'ensemble de l'image. Passé cette limite (cf. Eq. (5.2)), un coefficient plus modéré permet de conserver des détails tels que ceux dans les contours ou de petits objets autrement perdus lors d'une fusion médiane (cf. Tab. 5.2). Cette séquence présentant un bruit modéré, on peut tirer parti d'un plus grand nombre de composantes pour retenir plus de précision.

L'image de la Fig. 5.4 provient d'une séquence en noir et blanc dont l'enregistrement a été perturbé/déformé par un phénomène optique dû à la déviation des faisceaux lumineux par des superpositions de couches d'air de températures différentes créant des *effets de mirage* comme ceux créés dans les déserts. La séquence d'images présente ainsi beaucoup de bruit ce qui complexifie la détection des zones mobiles/immobiles par un algorithme utilisant une approche de soustraction d'arrière-plan en

Turbulence1		
$C = 4$	$\beta = 0.90$	$FM = 0.658$
$C = 4$	$\beta = 1.50$	$FM = 0.671$
$C = 4$	$\beta = 4.00$	$FM = 0.673$
$C = 4$	médiane	$FM = 0.684$
$C = 6$	$\beta = 0.90$	$FM = 0.654$
$C = 6$	$\beta = 1.50$	$FM = 0.666$
$C = 6$	$\beta = 4.00$	$FM = 0.673$
$C = 6$	médiane	$FM = 0.678$
GMM cd.net		$FM = 0.312$
GMM encapsulé		$FM = 0.490$

Boats		
$C = 6$	$\beta = 0.90$	$FM = 0.858$
$C = 6$	$\beta = 1.50$	$FM = 0.854$
$C = 6$	$\beta = 4.00$	$FM = 0.850$
$C = 6$	médiane	$FM = 0.838$
$C = 12$	$\beta = 0.90$	$FM = 0.847$
$C = 12$	$\beta = 1.50$	$FM = 0.861$
$C = 12$	$\beta = 4.00$	$FM = 0.850$
$C = 12$	médiane	$FM = 0.843$
GMM cd.net		$FM = 0.690$
GMM encapsulé		$FM = 0.690$

TABLE 5.2: *F-Measure FM des séquences résultantes de différentes fusions des séquences* TURBULENCE1 *et* BOATS*. En fonction du nombre de composantes C et du coefficient de fusion β, en comparaison aux résultats de l'algorithme GMM sur ChangeDetection.net et et de GMM paramétré tel qu'utilisé par notre application (cf Chapitre 6) .*

coordonnée cartésienne. Pour notre modèle, ce type de séquence d'images requiert l'utilisation d'un coefficient plus élevé et éventuellement un nombre de composantes plus réduit pour filtrer ce type de bruit ou de distorsion. Dans notre cas, on note alors une progression dans la suppression des faux positifs, mais elle reste négligeable (cf. Tab. 5.2, Fig. 5.5).

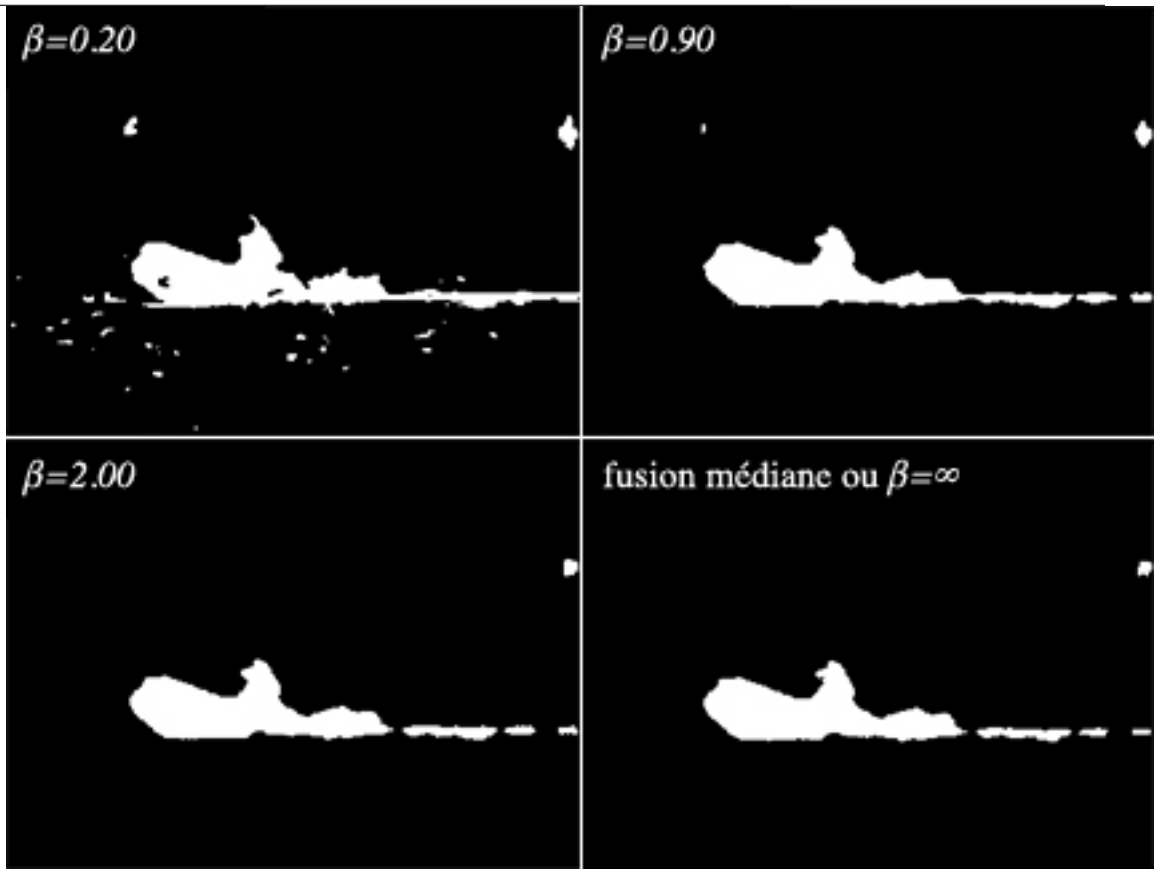

FIGURE 5.3: *Résultat des fusions des composantes (cf. Fig 3.14.) selon différents coefficients.*

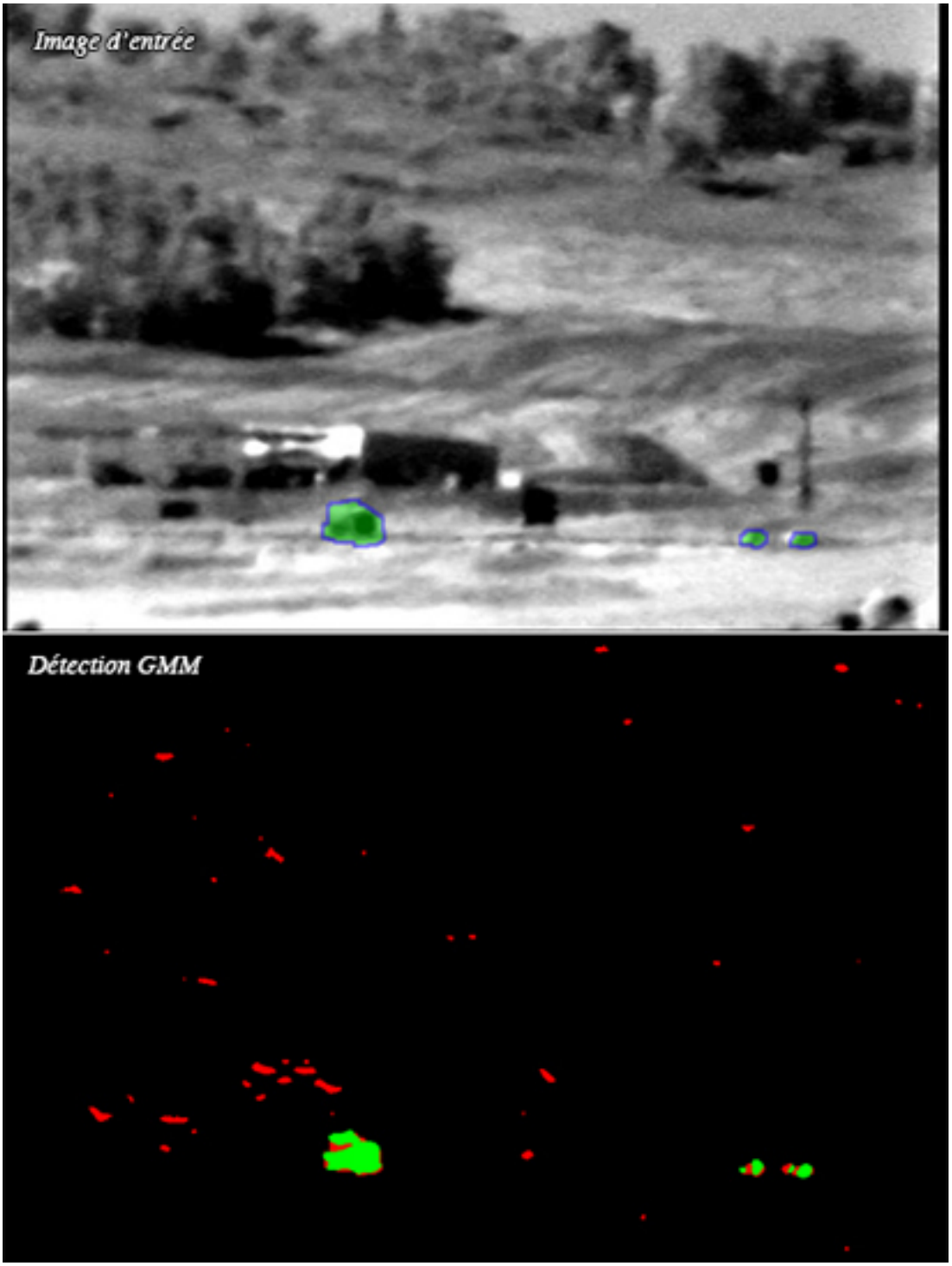

FIGURE 5.4: *En haut : Image* 1660 *issue de la séquence* TURBULENCE1*, la zone de mouvement est marquée en vert et la zone de tolérance en bleu. En bas : détection originale obtenue via GMM, en vert les succès et en rouge les échecs.*

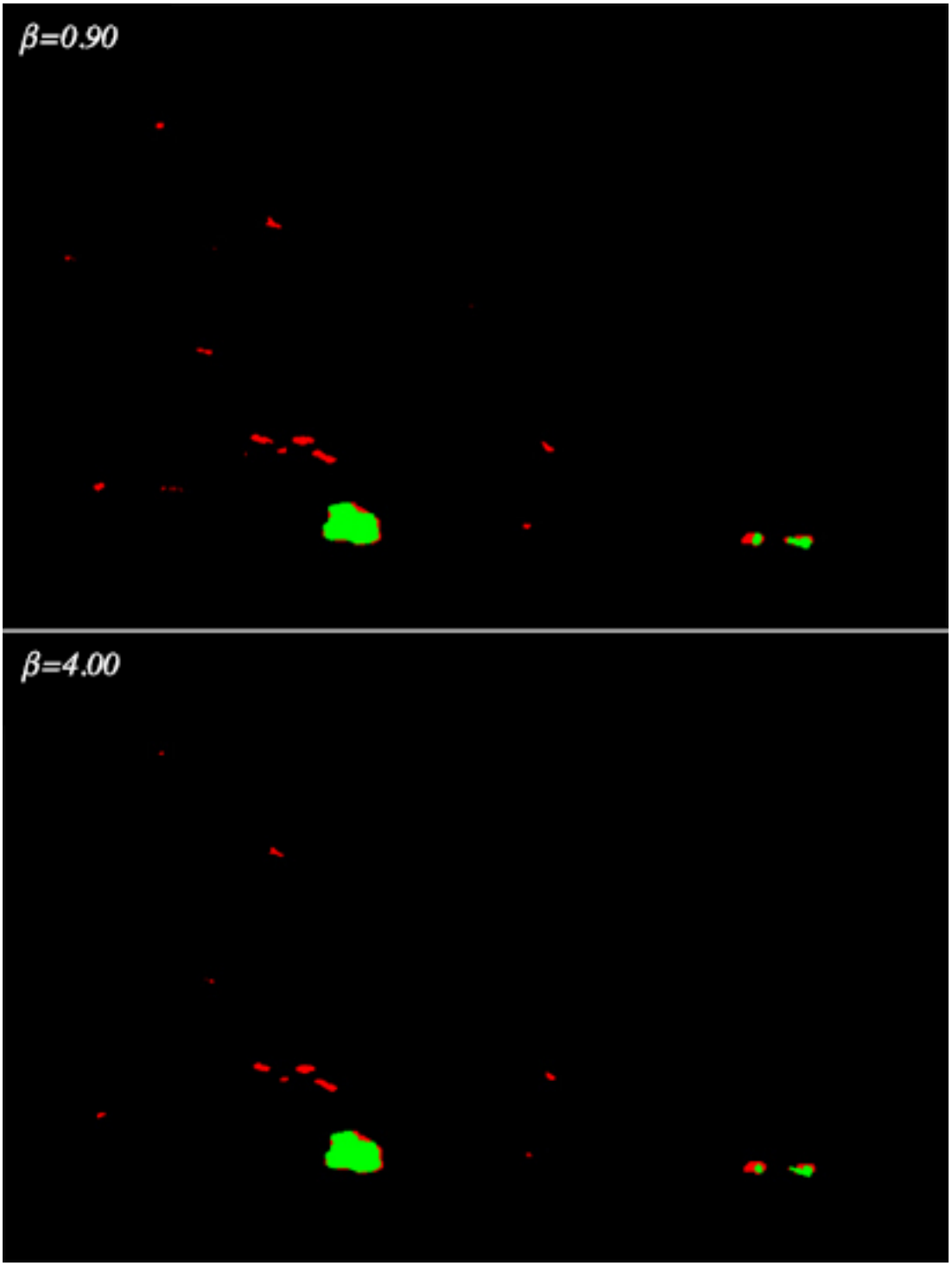

FIGURE 5.5: *Résultats obtenus après fusion selon différents coefficients d'une détection à 6 composantes, en vert les succès et en rouge les échecs.*

5.5 Conclusion

La multiplication des points d'intérêt de l'oeil virtuel (ou composantes), et leur pondération respective dans la fusion, permettent de conserver les détails tout en évitant le bruit présent dans la détection originale. En accord avec le procédé d'intégration trans-saccadique, il est mis en évidence l'intérêt de la fusion de différents points de vue autour d'un même point d'intérêt, copiant ainsi le processus du SVH d'intégration d'indices allocentriques et égocentriques à la rétine pour la perception d'un même point [17]. Cet algorithme utilisé sur des séquences de résolution plus importante sera employé avec proportionnellement plus de centres de transformation. Dans ce cas, lors de la fusion en chaque pixel, les points de vue les moins pondérés pourraient être ignorés pour gagner en temps de calcul.

Chapitre 6

IMPLÉMENTATION ET RÉSULTATS

6.1 F-Measure

Pour classer par ordre de succès les algorithmes de détection de mouvements présentés sur le site Changedetection.net, on a l'habitude de mesurer l'exactitude de leurs résultats au sens de la F-Mesure. Ces algorithmes fournissent une détection de mouvement sur l'ensemble des séquences vidéo de la base de données. Chaque image des séquences vidéo est associée à une vérité-terrain (*ground truth*) générée manuellement (cf. Fig. 6.1) qui représente la détection idéale avec laquelle est comparée la détection automatique des algorithmes à évaluer. La vérité-terrain affiche 4 intensités différentes selon le résultat attendu :

— 0 : région immobile.

— 85 : hors de la région d'intérêt, la détection automatique dans ces zones n'a pas d'impact dans le calcul de la F-mesure.

— 170 : inconnu, lorsque le contour d'un objet en mouvement est flou, notamment à cause d'un mouvement trop rapide. La détection automatique dans ces zones n'a pas d'impact dans le calcul de la F-mesure.

— 255 : région mobile.

La détection de mouvements étant essentiellement une classification binaire, l'évaluation implique les résultats possibles de détections suivants : positif, négatif, faux positif et faux négatifs. Ainsi on définit la précision et le rappel comme :

FIGURE 6.1: *Image de vérité terrain associé à l'image Fig. 1.1.*

$$\begin{aligned} \text{Précision} &= \frac{\text{positifs}}{\text{positifs} + \text{faux positif}} \\ \text{Rappel} &= \frac{\text{positifs}}{\text{positifs} + \text{faux négatif}} \end{aligned} \tag{6.1}$$

La F-Measure, ou score F, donne une idée de l'exactitude de la détection en combinant la précision et le rappel et en proposant une valeur comprise entre 0 à 1 (1 pour une classification parfaite) :

$$\text{F-Measure} = 2 * \frac{\text{Précision} * \text{Rappel}}{\text{Précision} + \text{Rappel}} \tag{6.2}$$

6.2 Paramétrage de GMM

Il est tout d'abord important de rappeler que les algorithmes testés sur CD-NET doivent être non supervisés (*automatiques*) et doivent ainsi utiliser les mêmes paramètres internes pour toutes les séquences vidéos de la base CDNET. D'autre part, comme l'algorithme GMM traite de la problématique d'extraction de fond indépendamment entre chaque pixel (cf. Chapitre 2), la transformation topologique opérée par notre algorithme représentant l'image en coordonnées logs-polaires ne devrait pas occasionner une modification de ses paramètres internes optimaux. Comme il a été expliqué dans le chapitre 2, les paramètres internes optimaux de l'algorithme GMM de Zivkovic [12] ont été préalablement reglés afin d'obtenir un bon taux de classification (*via* le F-mesure) tandis que les autres paramètres de cet algorithme sont automatisés et sont censés s'adapter dynamiquement durant son exécution. Notre expérience inclue ainsi cette même contrainte, pour les paramètres de l'algorithme GMM qui sera encapsulé dans notre modèle, comme pour les paramètres propres à notre algorithme, tel que le nombre de transformations impliquées. Les six paramètres à définir dans l'algorithme fourni par Zivkovic sont à l'origine définis comme :

```
1    // 1) K - max number of Gaussians per pixel
     pGMM->nM = 4; septs
     // 2) Tb - the threshold - n var
     pGMM->fTb = 4*4;
     // 3) Tbf - the threshold
6    pGMM->fTB = 0.90f;//1-cf from the paper
     // 4) Tgenerate - the threshold
     pGMM->fTg = 3.0f*3.0f; //update the mode or generate new
     pGMM->fSigma= 11.0f; //sigma for the new mode
     // 5) alpha - the learning factor
11   pGMM->fAlphaT=0.001f;
     // 6) complexity reduction prior constant
     pGMM->fCT=0.05f;
```

Or le site ChangeDetection.net désigne seulement pour cet algorithme les paramètres tels que " Default. K = 3, alpha = 0.001 " [1]. Les paramètres par défaut comme ceux apportés par CDNET n'ont pas permis d'obtenir les résultats mentionnés sur le site. Un échange de mails avec Pierre-Marc Jodoin (Université de Sherbrooke, organisateur de CDNET) et Nil Goyette (Université de Sherbrooke, ancien webmaster, programmeur) nous a apporté les paramètres dans leur ensemble, mais ceux-ci n'ont pas généré les résultats escomptés. Une recherche exhaustive de paramètres sur une sélection de vidéos choisies dans différentes catégories sur CDNET pour leur variété nous a permis de réduire les différences entre les résultats obtenus et ceux de CDNET. L'expérience nous fait conclure que la principale raison de différences entre les résultats était due à la compression et décompression des images au format JPEG fourni par le site.

Les paramètres de la recherche retenus pour notre expérience sont :

```
    //set parameters
2   // 1) K - max number of Gaussians per pixel
    pGMM->nM = 4;
    // 2) Tb - the threshold - n var
    pGMM->fTb = 4*4;
    // 3) Tbf - the threshold
7   pGMM->fTB = 0.90f;//1-cf from the paper
    // 4) Tgenerate - the threshold
    pGMM->fTg = 4.0f*4.0f;//update the mode or generate new
    pGMM->fSigma= 50.0f;//sigma for the new mode
    // 5) alpha - the learning factor
12  pGMM->fAlphaT=0.001f;
    // 6) complexity reduction prior constant
    pGMM->fCT=0.05f;
```

Avoir les mêmes résultats que CDNET à l'utilisation de GMM nous aurait permis une comparaison directe avec notre propre algorithme encapsulant GMM. Dans cette étude, nous nous en tiendrons à une comparaison de notre algorithme avec les résultats

1. http ://wordpress-jodoin.dmi.usherb.ca/method/96/

de notre utilisation de GMM avec les paramètres internes que nous avons jugés et considéré nous même comme optimaux pour la base CDNET.

6.3 Comparaison avec les résultats de la méthode de base GMM de Zivkovic

	GMM [1]	Résultats	Gains
BOULEVARD	0.571	0.563	−0.008
TRAMCROSSROAD_1FPS	0.852	0.855	0.003
HIGHWAY	0.904	0.921	0.017
OFFICE	0.379	0.406	0.027
CANOE	0.875	0.920	0.045
WETSNOW	0.585	0.718	0.133
PARK	0.708	0.794	0.85
TURBULENCE1	0.490	0.648	0.158
BOATS	0.670	0.858	0.187

TABLE 6.1: *Comparaison des résultats obtenus sur les séquences avec GMM et avec notre algorithme encapsulant GMM*

Les résultats sur la plupart des vidéos présentent des gains allant de faible à notable au sens de la F-measure pour notre stratégie de segmentation des zones mobiles en log-polaire et encapsulant GMM (cf. Tab. 6.1). Dans le pire des cas le score de détection de mouvement reste sensiblement le même. On note que les vidéos présentant originalement une détection de bonne qualité présentent un gain de F-measure moindre, ce qui semble logique. Les détections de très mauvaise qualité tendent aussi à ne pas pouvoir être améliorées par notre méthode.

6.3.1 Filtre médian

Les résultats des algorithmes présentés sur CDNET sont sujets à un filtre médian de $5 * 5$ pixels antérieurs au calcul de la F-Measure, réduisant ainsi la majeure partie du bruit à la détection.

6.4 Conclusion

Pour calculer le gain, les résultats de l'algorithme GMM [1] sont soustraits à ceux obtenus par notre algorithme (cf. Tab. 6.1) car ce dernier encapsule le premier. Ce gain étant calculé par rapport aux résultats de l'algorithme GMM [1] filtrés par un filtre médian (tel que décrit dans le paragraphe précédent), on peut en conclure que notre algorithme fournit une amélioration de la détection de mouvement par rapport au simple filtre médian.

Chapitre 7

CONCLUSION

Ce mémoire de maîtrise a permis de mettre en avant une technique utilisée par le SVH, c'est-à-dire l'utilisation de la transformée log-polaire et la fusion de différents points de vue obtenus dans cet espace comme stratégie intéressante pour améliorer un algorithme de détection de mouvements. Nous avons pu constater que l'utilisation de cette stratégie rendait plus robuste cet algorithme de détection de zones mobiles face à la grande variété de scènes pouvant être filmée et représentée dans la CDNET. Ainsi, nous pensons que la stratégie décrite dans ce mémoire pourrait être éventuellement adaptée et combinée avec d'autres algorithmes de traitement d'images et de vision par ordinateur afin d'améliorer, *in fine*, leur robustesse.

Bien que beaucoup d'algorithmes de détection de mouvement ne sont que des variantes de l'algorithme GMM de Zivkovic (cf. chapitre 2), il serait intéressant d'étudier les résultats de notre stratégie sur un autre algorithme de détection de fond, particulièrement pour les modèles proposés dans la littérature qui ne se restreigne pas au niveau du pixel et qui accorderait une plus grande importance aux transformations topologiques opérées avant la détection.

La construction de notre algorithme divise l'image originale en plusieurs transformées et permet ainsi de repartir les calculs qui y sont appliqués sur plusieurs processeurs et gagner en temps de calcul. De plus, les images log-polaire contenant moins d'informations que l'image source, elles sont ainsi individuellement plus légères en mémoire que l'image originale.

Nous avons vu dans le chapitre 3 que les capteurs de caméras, contrairement

à la rétine, sont généralement organisés sur un repère cartésien, cependant [19,20] font l'utilisation de caméras log-polaire (cf. Fig. 7.1), évitant ainsi le temps de calcul associé à la transformation log-polaire. Notre système pourrait s'adapter à l'utilisation de multiples caméras de ce type.

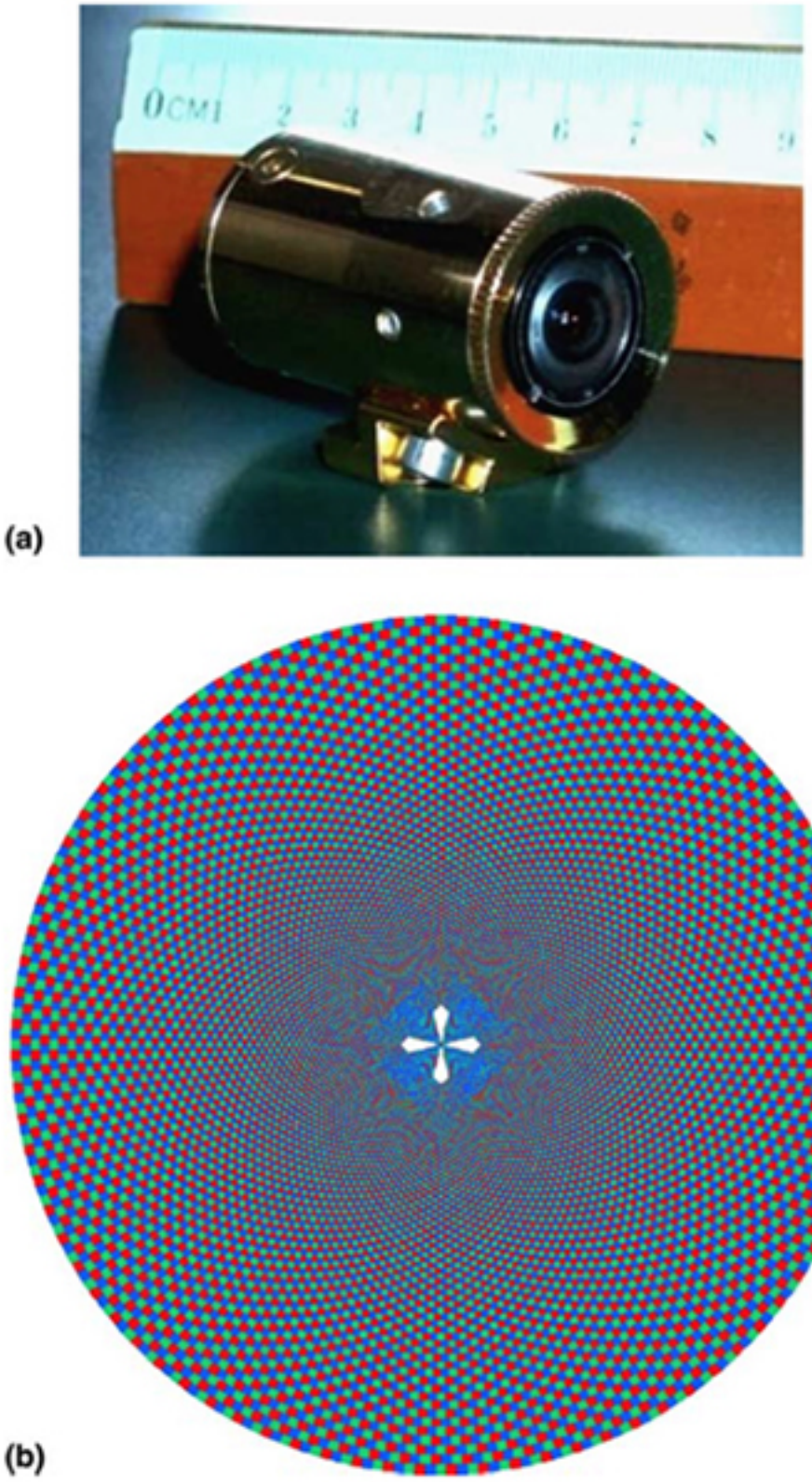

FIGURE 7.1: *Issu de [9]. En haut : caméra log-polaire. En bas : arrangement des pixels du capteur.*

RÉFÉRENCES

[1] Zoran Zivkovic : Improved adaptative gaussian mixture model for background subtraction. *Intelligent And Autonomous System Group*, 2004.

[2] JK Bowmaker et H.J.A. Dartnall : Visual pigments of rods and cones in a human retina. *M.R.C. Vision Unit*, 1979.

[3] Dale Purves, George J Augustine, David Fitzpatrick, Lawrence C Katz, Anthony-Samuel LaMantia, James O McNamara et S Mark Williams, éditeurs. *Neuroscience*, volume 2, chapitre 11-20. Sinauer Associates, 2001.

[4] TEST : Test. *TEST*, 14:989–1006, 2004.

[5] H. Araujo et J. Dias : An introduction to the log-polar mapping. *Proceedings on second Workshop on Cybernetic Vision and IEEE*, page 139–144, 1996.

[6] Max Mignotte : A biologically-inspired framework for contour detection. *Springer-Verlag London*, 2015.

[7] Xiangui Kang, Jiwu Huang et Wenjun Zeng : Efficient general print-scanning resilient data hiding based on uniform log-polar mapping. *EEE Transactions on Information Forsenics and Security*, 5(1), 2010.

[8] R. Monzotti, A. Gasteratos, G. Metta et G. Sandini : Disparity estimation on log-polar images and vergence control. *Computer Vision and Image Understanding*, 83:87–117, 2001.

[9] Giorgio Metta, Antonios Gasteratos et Gulio Sandini : Learning to track colored objects with log-polar vision. *Mechatronics*, 14:989–1006, 2004.

[10] Weiman et Chaikin : Logarithmic spiral grids for image processing and display. *Computer Graphics and Image Processing*, 11:197–226, 1979.

[11] Chris Stauffer et W.E.L Grimson : Adaptive background mixture models

for real-time tracking. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 246–242, 1999.

[12] Z.Zivkovic et F.van der Heijden : Recursive unsupervised learning of finite mixture models. *IEEE Transaction on PAMI*, 26(5), 2004.

[13] D. Coppola et D. Purves : The extraordinarily rapid disappearance of entopic images. *Proceedings of the National Academy of science of the United States of America*, 1996.

[14] Takio Kurita, Kazuhiro Hotta et Taketoshi Mishima : Scale and rotation invariant recognition method using high-order local autocorrelation features of log-polar image. *Asian Conference on Computer Vision*, 3, 1998.

[15] Chi-Man Pun et Moon-Chuen Lee : Log-polar wavelet energy signatures for rotation and scale invariant texture classification. *IEEE transactions on pattern analysis and machine intelligence*, 25(5):590, 2003.

[16] Marc Bolduc et Martin D. Levine : A review of biologically motivated space variant data reduction models for robotic vision. *Computer Vision and Image Understanding (CVIU)*, 1996.

[17] Steven L. Prime, Michael Vesia et J. Douglas Crawford : Cortical mechanisms for trans-saccadic memory and integration of multiple object features. *Phil. Trans. R. Soc. B*, pages 540–543, 2011.

[18] Tamersoy : Background subtraction lecture notes, 2009.

[19] George Wolberg et Siavash Zokai : Robust image registration using log-polar transform. *Proceedings of IEEE International Conference on Image Processing*, 2000.

[20] George Wolberg et Siavash Zokai : Image registration using log-polar mappings for recovery of large scale similarity and projective transformations. *IEEE Transactions on Image Processing*, 14(10), 2005.

[21] Shelby Pereira, Joseph J. K. O Ruanaidh, Frédéric Deguillaume, Gabriela Csurka et Thierry Pun : Template based recovery of fourier-based watermarks using log-polar and log-log maps. *IEEE Int Conf onn Multimedia Computing and Systems (ICMCS)*, 1999.

[22] B.E. Bayer : Color imaging array, juillet 20 1976. US Patent 3,971,065.

ANNEXE A : IMPLÉMENTATION DU MODÈLE DE MÉLANGE DE GAUSSIENNES DE Z.ZIVKOVIK

7.2 CvPixelBackgroundGMM.h

```
//Implementation of the Gaussian mixture model background subtraction from:
//"Improved adaptive Gausian mixture model for background subtraction" Z.Zivkovic
    International Conference Pattern Recognition, UK, August, 2004 and "Efficient
    Adaptive Density Estimapion per Image Pixel for the Task of Background
    Subtraction" Z.Zivkovic, F. van der Heijden Pattern Recognition Letters, vol.
    27, no. 7, pages 773-780, 2006. The algorithm similar to the standard Stauffer&
    Grimson algorithm with additional selection of the number of the Gaussian
    components based on: "Recursive unsupervised learning of finite mixture models "
     Z.Zivkovic, F.van der Heijden IEEE Trans. on Pattern Analysis and Machine
    Intelligence, vol.26, no.5, pages 651-656, 2004

//////////
//Author: Z.Zivkovic, www.zoranz.net
//University of Amsterdam, The Netherlands
//Date: 27-April-2005, Version:0.9
//////////

#include <stdlib.h>
#include <memory.h>

typedef struct CvPBGMMGaussian
{
  float sigma;
  float muR;
  float muG;
  float muB;
  float weight;
}CvPBGMMGaussian;

typedef struct CvPixelBackgroundGMM
{
  ////////////////////////
  //very important parameters - things you will change
```

```
26  ////////////////////////
    float fAlphaT;
    //alpha - speed of update - if the time interval you want to average over is T set
        alpha=1/T. It is also usefull at start to make T slowly increase from 1 until
        the desired T
    float fTb;
    //Tb - threshold on the squared Mahalan. dist. to decide if it is well described
        by the background model or not. Related to Cthr from the paper. This does not
        influence the update of the background. A typical value could be 4 sigma and
        that is Tb=4*4=16;
31
    ////////////////////////
    //less important parameters - things you might change but be carefull
    ////////////////////////
    float fTg;
36  //Tg - threshold on the squared Mahalan. dist. to decide when a sample is close to
        the existing components. If it is not close to any a new component will be
        generated. I use 3 sigma => Tg=3*3=9. Smaller Tg leads to more generated
        components and higher Tg might make lead to small number of components but
        they can grow too large
    float fTB;//1-cf from the paper
    //TB - threshold when the component becomes significant enough to be included into
        the background model. It is the TB=1-cf from the paper. So I use cf=0.1 => TB
        =0. For alpha=0.001 it means that the mode should exist for approximately 105
        frames before it is considered foreground
    float fSigma;
    //initial standard deviation  for the newly generated components. It will will
        influence the speed of adaptation. A good guess should be made. A simple way
        is to estimate the typical standard deviation from the images. I used here 10
        as a reasonable value
41  float fCT;
    //CT - complexity reduction prior this is related to the number of samples needed
        to accept that a component actually exists. We use CT=0.05 of all the samples.
        By setting CT=0 you get the standard Stauffer&Grimson algorithm (maybe not
        exact but very similar)

    //even less important parameters
    int nM;//max number of modes - const - 4 is usually enough
46
    //shadow detection parameters
    int bShadowDetection;//do shadow detection
    float fTau;
```

```
    // Tau - shadow threshold. The shadow is detected if the pixel is darker version
        of the background. Tau is a threshold on how much darker the shadow can be.
        Tau= 0.5 means that if pixel is more than 2 times darker then it is not shadow
         See: Prati,Mikic,Trivedi,Cucchiarra,"Detecting Moving Shadows...",IEEE PAMI
        ,2003.
51
    //data
    int nNBands;//only RGB now ==3
    int nWidth;//image size
    int nHeight;
56  int nSize;
    // dynamic array for the mixture of Gaussians
    CvPBGMMGaussian* rGMM;
    unsigned char* rnUsedModes;//number of Gaussian components per pixel
    bool bRemoveForeground;
61  } CvPixelBackgroundGMM;


    void cvUpdatePixelBackgroundGMM(CvPixelBackgroundGMM* pGMM,unsigned char* data,
        unsigned char* output);
    //Input:
66  //  pGMM - a pointer to an alrady initialized GMM
    //  data - a pointer to the data of a RGB image of the same size
    //Output:
    //  out - a pointer to the data of a gray value image of the same size (the memory
        should already be reserved)
    //      values: 255-foreground, 125-shadow, 0-background
71  ////////////
    void cvUpdatePixelBackgroundGMMTiled(CvPixelBackgroundGMM* pGMM,unsigned char* data,
        unsigned char* output);
    //Use in case R G B images are separate - e.g. calling from Matlab

    #if 1
76

    CvPixelBackgroundGMM* cvCreatePixelBackgroundGMM(int width,int height);

    void cvReleasePixelBackgroundGMM(CvPixelBackgroundGMM** ppGMM);

81  //this might be usefull
    void cvSetPixelBackgroundGMM(CvPixelBackgroundGMM* pGMM,unsigned char* data);
    #endif
```

7.3 CvPixelBackgroundGMM.cpp

```
//#include "stdafx.h"
#include "CvPixelBackgroundGMM.h"
CvPixelBackgroundGMM* cvCreatePixelBackgroundGMM(int width,int height)
{
  CvPixelBackgroundGMM* pGMM=new(CvPixelBackgroundGMM);
  int size=width*height;
  pGMM->nWidth=width;
  pGMM->nHeight=height;
  pGMM->nSize=size;
  pGMM->nNBands=3;//always 3 - not implemented for other values!
  //set parameters
  // K - max number of Gaussians per pixel
  pGMM->nM = 4;
  // Tb - the threshold - n var
  pGMM->fTb = 4*4;
  // Tbf - the threshold
  pGMM->fTB = 0.9f;//1-cf from the paper
  // Tgenerate - the threshold
  pGMM->fTg = 3.0f*3.0f;//update the mode or generate new
  pGMM->fSigma= 11.0f;//sigma for the new mode
  // alpha - the learning factor
  pGMM->fAlphaT=0.001f;
  // complexity reduction prior constant
  pGMM->fCT=0.05f;
  //shadow
  // Shadow detection
  pGMM->bShadowDetection = 1;//turn on
  pGMM->fTau = 0.5f;// Tau - shadow threshold
  //GMM for each pixel
  pGMM->rGMM=(CvPBGMMGaussian*) malloc(size * pGMM->nM * sizeof(CvPBGMMGaussian));
  //used modes per pixel
  pGMM->rnUsedModes = (unsigned char* ) malloc(size);
  memset(pGMM->rnUsedModes,0,size);//no modes used
    pGMM->bRemoveForeground=0;
  return pGMM;
}

void cvReleasePixelBackgroundGMM(CvPixelBackgroundGMM** ppGMM)
{
  delete (*ppGMM)->rGMM;
```

```
  delete (*ppGMM)->rnUsedModes;
  delete (*ppGMM);
  (*ppGMM)=0;
}

//this might be usefull
void cvSetPixelBackgroundGMM(CvPixelBackgroundGMM* pGMM, unsigned char* data)
{
  int size=pGMM->nSize;
  unsigned char* pDataCurrent=data;

  for (int i=0;i<size;i++)
  {
  // retrieve the colors
    float R = *pDataCurrent++;
    float G = *pDataCurrent++;
    float B = *pDataCurrent++;

    pGMM->rGMM[i].weight=1.0;
    pGMM->rGMM[i].muR=R;
    pGMM->rGMM[i].muG=G;
    pGMM->rGMM[i].muB=B;
    pGMM->rGMM[i].sigma=pGMM->fSigma;
  }


  memset(pGMM->rnUsedModes,1,size);//1 mode used
}

int _cvRemoveShadowGMM(long posPixel,
                       float red, float green, float blue,
                       unsigned char nModes,
                       CvPBGMMGaussian* m_aGaussians,
                       int m_nM,
                       float m_fTb,
                       float m_fTB,
                       float m_fTg,
                       float m_fTau)
{
  //calculate distances to the modes (+ sort???)
  //here we need to go in descending order!!!
//  long posPixel = pixel * m_nM;
```

```
    long pos;
    float tWeight = 0;
    float numerator, denominator;
    // check all the distributions, marked as background:
87  for (int iModes=0;iModes<nModes;iModes++)
    {
      pos=posPixel+iModes;
      float var = m_aGaussians[pos].sigma;
      float muR = m_aGaussians[pos].muR;
92    float muG = m_aGaussians[pos].muG;
      float muB = m_aGaussians[pos].muB;
      float weight = m_aGaussians[pos].weight;
      tWeight += weight;

97    numerator = red * muR + green * muG + blue * muB;
      denominator = muR * muR + muG * muG + muB * muB;
      // no division by zero allowed
      if (denominator == 0)
      {
102       break;
      };
      float a = numerator / denominator;
      // if tau < a < 1 then also check the color distortion
      if ((a <= 1) && (a >= m_fTau))//m_nBeta=1
107   {
        float dR=a * muR - red;
        float dG=a * muG - green;
        float dB=a * muB - blue;
        //square distance -slower and less accurate
112     //float maxDistance = cvSqrt(m_fTb*var);
        //if ((fabs(dR) <= maxDistance) && (fabs(dG) <= maxDistance) && (fabs(dB) <=
            maxDistance))
        //circle
        float dist=(dR*dR+dG*dG+dB*dB);
        if (dist<m_fTb*var*a*a)
117     {
          return 2;
        }

      };
122   if (tWeight > m_fTB)
      {
```

```
        break;
      };
    };
127 return 0;
}

int _cvUpdatePixelBackgroundGMM(long posPixel,
                float red, float green, float blue,
132             unsigned char* pModesUsed,
                CvPBGMMGaussian* m_aGaussians,
                int m_nM,
                float m_fAlphaT,
                float m_fTb,
137             float m_fTB,
                float m_fTg,
                float m_fSigma,
                float m_fPrune)
{
142 //calculate distances to the modes (+ sort???)
  //here we need to go in descending order!!!

  //long posPixel = pixel * m_nM;
  long pos;
147 //  long pos=posPixel-1;//because of ++ at the end

  bool bFitsPDF=0;
  bool bBackground=0;
  float m_fOneMinAlpha=1-m_fAlphaT;
152 //bool bPrune=0;
  int nModes=*pModesUsed;
  float totalWeight=0.0f;
  //////
  //go through all modes
157 for (int iModes=0;iModes<nModes;iModes++)
  {
    pos=posPixel+iModes;
    float weight = m_aGaussians[pos].weight;
    ////
162 //fit not found yet
    if (!bFitsPDF)
    {
      //check if it belongs to some of the modes
```

```
        //calculate distance
167     float var = m_aGaussians[pos].sigma;
        float muR = m_aGaussians[pos].muR;
        float muG = m_aGaussians[pos].muG;
        float muB = m_aGaussians[pos].muB;

172     float dR=muR - red;
        float dG=muG - green;
        float dB=muB - blue;
        ///////
        //check if it fits the current mode (Factor * sigma)
177
        //square distance -slower and less accurate
        //float maxDistance = cvSqrt(m_fTg*var);
        //if ((fabs(dR) <= maxDistance) && (fabs(dG) <= maxDistance) && (fabs(dB) <=
            maxDistance))
        //circle
182     float dist=(dR*dR+dG*dG+dB*dB);
        //background? - m_fTb
        if ((totalWeight<m_fTB)&&(dist<m_fTb*var))
            bBackground=1;
        //check fit
187     if (dist<m_fTg*var)
        {
          /////
          //belongs to the mode
          bFitsPDF=1;
192       //update distribution
          float k = m_fAlphaT/weight;
          weight=m_fOneMinAlpha*weight+m_fPrune;
          weight+=m_fAlphaT;
          m_aGaussians[pos].muR = muR - k*(dR);
197       m_aGaussians[pos].muG = muG - k*(dG);
          m_aGaussians[pos].muB = muB - k*(dB);
          //limit update speed for cov matrice
          //not needed
          //k=k>20*m_fAlphaT?20*m_fAlphaT:k;
202       //float sigmanew = var + k*((0.33*(dR*dR+dG*dG+dB*dB))-var);
          //float sigmanew = var + k*((dR*dR+dG*dG+dB*dB)-var);
          //float sigmanew = var + k*((0.33*dist)-var);
          float sigmanew = var + k*(dist-var);
          //limit the variance
```

```
207         //m_aGaussians[pos].sigma = sigmanew>70?70:sigmanew;
            //m_aGaussians[pos].sigma = sigmanew>5*m_fSigma?5*m_fSigma:sigmanew;
            m_aGaussians[pos].sigma =sigmanew< 4 ? 4 : sigmanew>5*m_fSigma?5*m_fSigma:
                sigmanew;
            //m_aGaussians[pos].sigma =sigmanew< 4 ? 4 : sigmanew>3*m_fSigma?3*m_fSigma:
                sigmanew;
            //m_aGaussians[pos].sigma = m_fSigma;
212         //sort
            //all other weights are at the same place and
            //only the matched (iModes) is higher -> just find the new place for it
            for (int iLocal = iModes;iLocal>0;iLocal--)
            {
217           long posLocal=posPixel + iLocal;
              if (weight < (m_aGaussians[posLocal-1].weight))
              {
                break;
              }
222
              else
              {
                //swap
                CvPBGMMGaussian temp = m_aGaussians[posLocal];
227             m_aGaussians[posLocal] = m_aGaussians[posLocal-1];
                m_aGaussians[posLocal-1] = temp;
              }

            }
232
            //belongs to the mode
            /////
          }

237       else
          {
            weight=m_fOneMinAlpha*weight+m_fPrune;
            //check prune
            if (weight<-m_fPrune)
242         {
              weight=0.0;
              nModes--;
            //  bPrune=1;
              //break;//the components are sorted so we can skip the rest
```

```
247         }

          }

          //check if it fits the current mode (2.5 sigma)
252       ///////
        }

        //fit not found yet
        /////
257     else
        {
            weight=m_fOneMinAlpha*weight+m_fPrune;
            //check prune
            if (weight<-m_fPrune)
262         {
              weight=0.0;
              nModes--;
              //bPrune=1;
              //break;//the components are sorted so we can skip the rest
267         }

        }

        totalWeight+=weight;
272     m_aGaussians[pos].weight=weight;
      }

      //go through all modes
      //////
277   //renormalize weights
      for (int iLocal = 0; iLocal < nModes; iLocal++)
      {
        m_aGaussians[posPixel+ iLocal].weight = m_aGaussians[posPixel+ iLocal].weight/
            totalWeight;
      }
282


      //make new mode if needed and exit
      if (!bFitsPDF)
      {
287     if (nModes==m_nM)
```

```
    {
            //replace the weakest
    }

292 else
    {
            //add a new one
      //totalWeight+=m_fAlphaT;
      //pos++;
297   nModes++;
    }

    pos=posPixel+nModes-1;
        if (nModes==1)
302   m_aGaussians[pos].weight=1;
    else
      m_aGaussians[pos].weight=m_fAlphaT;
    //renormalize weights
    int iLocal;
307 for (iLocal = 0; iLocal < nModes-1; iLocal++)
    {
      m_aGaussians[posPixel+ iLocal].weight *=m_fOneMinAlpha;
    }

312 m_aGaussians[pos].muR=red;
    m_aGaussians[pos].muG=green;
    m_aGaussians[pos].muB=blue;
    m_aGaussians[pos].sigma=m_fSigma;
    //sort
317 //find the new place for it
    for (iLocal = nModes-1;iLocal>0;iLocal--)
    {
      long posLocal=posPixel + iLocal;
      if (m_fAlphaT < (m_aGaussians[posLocal-1].weight))
322   {
            break;
      }

      else
327   {
        //swap
        CvPBGMMGaussian temp = m_aGaussians[posLocal];
```

```
          m_aGaussians[posLocal] = m_aGaussians[posLocal-1];
          m_aGaussians[posLocal-1] = temp;
332       }


      }


  }
337
  //set the number of modes
  *pModesUsed=nModes;
    return bBackground;
}
342
void _cvReplacePixelBackgroundGMM(long pos,
                unsigned char* pData,
                CvPBGMMGaussian* m_aGaussians)
{
347  pData[0]=(unsigned char) m_aGaussians[pos].muR;
  pData[1]=(unsigned char) m_aGaussians[pos].muG;
  pData[2]=(unsigned char) m_aGaussians[pos].muB;
}

352 void cvUpdatePixelBackgroundGMM(CvPixelBackgroundGMM* pGMM, unsigned char* data,
    unsigned char* output)
{
  int size=pGMM->nSize;
  unsigned char* pDataCurrent=data;
  unsigned char* pUsedModes=pGMM->rnUsedModes;
357  unsigned char* pDataOutput=output;
  //some constants
  int m_nM=pGMM->nM;
  float m_fAlphaT=pGMM->fAlphaT;
  float m_fTb=pGMM->fTb;//Tb - threshold on the Mahalan. dist.
362  float m_fTB=pGMM->fTB;//1-TF from the paper
  float m_fTg=pGMM->fTg;//Tg - when to generate a new component
  float m_fSigma=pGMM->fSigma;//initial sigma
  float m_fCT=pGMM->fCT;//CT - complexity reduction prior
  float m_fPrune=-m_fAlphaT*m_fCT;
367  float m_fTau=pGMM->fTau;
  CvPBGMMGaussian* m_aGaussians=pGMM->rGMM;
  long posPixel;
  int m_bShadowDetection=pGMM->bShadowDetection;
```

```
    //go through the image
372 for (int i=0;i<size;i++)
    {
      // retrieve the colors
      float red = *pDataCurrent++;
      float green = *pDataCurrent++;
377   float blue = *pDataCurrent++;

      //update model+ background subtract
      ;
      posPixel=i*m_nM;
382   int result = _cvUpdatePixelBackgroundGMM(posPixel, red, green, blue,pUsedModes,
          m_aGaussians,
        m_nM,m_fAlphaT, m_fTb, m_fTB, m_fTg, m_fSigma, m_fPrune);
      int nMLocal=*pUsedModes;
      pUsedModes++;
      if (m_bShadowDetection)
387       if (!result)
          {
            result= _cvRemoveShadowGMM(posPixel, red, green, blue,nMLocal,m_aGaussians
                ,
                  m_nM,
                  m_fTb,
392               m_fTB,
                  m_fTg,
                  m_fTau);
          }

397

      switch (result)
      {
        case 0:
          //foreground
402       (* pDataOutput)=255;
          if (pGMM->bRemoveForeground)
          {
            _cvReplacePixelBackgroundGMM(posPixel,pDataCurrent-3,m_aGaussians);
          }
407
          break;
        case 1:
          //background
```

```
          (* pDataOutput)=0;
412       break;
        case 2:
          //shadow
          (* pDataOutput)=125;
          if (pGMM->bRemoveForeground)
417       {
            _cvReplacePixelBackgroundGMM(posPixel,pDataCurrent-3,m_aGaussians);
          }

          break;
422     }

      //(* pDataOutput)=nM*30;
      pDataOutput++;
    }
427
  }

  void cvUpdatePixelBackgroundGMMTiled(CvPixelBackgroundGMM* pGMM,unsigned char* data,
      unsigned char* output)
  {
432   int size=pGMM->nSize;
    unsigned char* pDataCurrentR=data;//separate R G and B images
    unsigned char* pDataCurrentG=data+size;
    unsigned char* pDataCurrentB=data+2*size;
    unsigned char* pUsedModes=pGMM->rnUsedModes;
437 unsigned char* pDataOutput=output;
    //some constants
    int m_nM=pGMM->nM;
    float m_fAlphaT=pGMM->fAlphaT;
    float m_fTb=pGMM->fTb;//Tb - threshold on the Mahalan. dist.
442 float m_fTB=pGMM->fTB;//1-TF from the paper
    float m_fTg=pGMM->fTg;//Tg - when to generate a new component
    float m_fSigma=pGMM->fSigma;//initial sigma
    float m_fCT=pGMM->fCT;//CT - complexity reduction prior
    float m_fPrune=-m_fAlphaT*m_fCT;
447 float m_fTau=pGMM->fTau;
    CvPBGMMGaussian* m_aGaussians=pGMM->rGMM;
    long posPixel;
    int m_bShadowDetection=pGMM->bShadowDetection;
    //go through the image
```

```
452   for (int i=0;i<size;i++)
      {
        // retrieve the colors
        float red = *pDataCurrentR++;
        float green = *pDataCurrentG++;
457     float blue = *pDataCurrentB++;

        //update model+ background subtract
        ;
        posPixel=i*m_nM;
462     int result = _cvUpdatePixelBackgroundGMM(posPixel, red, green, blue,pUsedModes,
            m_aGaussians,
          m_nM,m_fAlphaT, m_fTb, m_fTB, m_fTg, m_fSigma, m_fPrune);
        int nMLocal=*pUsedModes;
        pUsedModes++;
        if (m_bShadowDetection)
467         if (!result)
            {
              result= _cvRemoveShadowGMM(posPixel, red, green, blue,nMLocal,m_aGaussians
                  ,
                    m_nM,
                    m_fTb,
472                 m_fTB,
                    m_fTg,
                    m_fTau);
            }


477
        switch (result)
        {
          case 0:
            //foreground
482         (* pDataOutput)=255;
            if (pGMM->bRemoveForeground)
            {
//              _cvReplacePixelBackgroundGMM(posPixel,pDataCurrent-3,m_aGaussians);
            }
487
            break;
          case 1:
            //background
            (* pDataOutput)=0;
```

```
492             break;
            case 2:
              //shadow
              (* pDataOutput)=125;
              if (pGMM->bRemoveForeground)
497           {
//                _cvReplacePixelBackgroundGMM(posPixel,pDataCurrent-3,m_aGaussians);
              }

              break;
502         }

          //(* pDataOutput)=nM*30;
          pDataOutput++;
        }
507
}
```

ANNEXE B : ALGORITHMES

7.4 Algorithme de pré-traitement et post-traitement

```
VideoFolder videoFolder(path_inputDir, path_inputDir+"output/");
videoFolder.readTemporalFile();
videoFolder.setExtension(".pgm");
GetLengthWidth(videoFolder.inputFrame(1).c_str(), &length, &width);
Comparator comparator(videoFolder);

//nombre de composantes
int nbLogDef = 6;

CvPixelBackgroundGMM* pGMM=0;
float *** matImgInput;
float ** resultSeg;
float ** gtFrame;

//initialisation de la methode de soustraction de fond individuellement pour chaque
    future composante
pGMM = new CvPixelBackgroundGMM[nbLogDef];
for(int l = 0; l < nbLogDef; l++) {
    pGMM[l] = *cvCreatePixelBackgroundGMM(width,length);
}

//boucle sur chaque image de la sequence
int temporalROIStart = videoFolder.getRange().first;
int temporalROIEnd = videoFolder.getRange().second;
for(int t = 1; t <= temporalROIEnd; t++) {
    //chargement de l'image
    matImgInput = fmatrix_allocate_3d(3, length, width);
    resultSeg = fmatrix_allocate_2d(length, width);
    gtFrame = fmatrix_allocate_2d(length, width);
    load_image_ppm(videoFolder.inputFrame(t).c_str(), matImgInput, length, width);

    //transformation log-polaire
    ImageTransform trans(matImgInput, length, width, nbLogDef);
    trans.TransfLogPolar();
```

```
    //applique la soustraction de fond sur chaque transformee log-polaire de l'image
    for(int d = 0; d < nbLogDef; d++) {
        cvUpdatePixelBackgroundGMM(&pGMM[d],trans.ImgTab1[d],trans.ImgTab1_1d[d]);
    }
    if(t >= temporalROIStart) {
        for(int d = 0; d < nbLogDef; d++) {
            MedianFilter(trans.ImgTab1_1d[d], length, width, 3);
            expand(trans.ImgTab1_1d[d], length, width, 3, true);
            FuseSmallRegionsBin(trans.ImgTab1_1d[d], length, width, 15);
            contract(trans.ImgTab1_1d[d], length, width, 3, true);
        }

        //transformation log-polaire inverse et fusion en une carte de detection
        trans.TransfInverseLogPolar();
        crossResults(trans.ImgTab2_1d, resultSeg, length, width, 3, nbLogDef, 0.90f)
            ;

        //calcule la fmeasure courrante
        load_image_pgm((char*)videoFolder.gtFrame(t).c_str(), gtFrame, length, width
            );
        comparator.compare(resultSeg, gtFrame);
        if(t%200==0) cout << t << " FM : " << comparator.getFMeasure() << endl;

        //sauvegarde le resultat
        ostringstream oss;
        oss << setfill('0') << setw(6) << t << ".pgm";
        SaveImagePgm((char*)videoFolder.getBinaryPath().c_str(), (char*)oss.str().
            c_str(), resultSeg, length, width);
    }

    //Free memory
    free_fmatrix_3d(matImgInput, 3);
    free_fmatrix_2d(resultSeg);
    free_fmatrix_2d(gtFrame);
}

cout << comparator.getFMeasure() << endl << endl << endl;
```

7.5 Algorithme de fusion des composantes

```
void crossResults(float*** mtSeg_tab, float ** mt_out, int lgth,int wdth,int SzN,
    int nbdef, float paramA) {
    int d,i,j,ii,jj;
    float pixelWhite;
    float pixelBlack;

    int** TabPtsInc = imatrix_allocate_2d(nbdef, 2);

    int nbRows = 2;
    int nbCols = 3;
    int size = wdth/(nbCols+1);
    int rowStart = (lgth-(nbRows-1)*size)/2;

    for(int r=0; r<nbRows; r++) for(int c=0; c<nbCols; c++) {
        TabPtsInc[r*(int)nbCols+c][0]=rowStart+r*size;
        TabPtsInc[r*(int)nbCols+c][1]=(1+c)*size;
    }

    int distMin = wdth/(nbCols+1);

    for(i=0;i<lgth;i++) for(j=0;j<wdth;j++) {
        pixelWhite = 0;
        pixelBlack = 0;
        for(ii=-(SzN/2); ii<=(SzN/2); ii++) for(jj=-(SzN/2); jj<=(SzN/2); jj++) //
            median filer
            if(sqrt(pow(ii, 2)+pow(jj, 2))<=(SzN/2) && i+ii>=0 && i+ii<lgth && j+jj
                >=0 && j+jj<wdth) {
                for(d=0;d<nbdef;d++) {
                    //distance from transform origin
                    float dist = sqrt(pow((TabPtsInc[d][0]-(i+ii)), 2)+pow((
                        TabPtsInc[d][1]-(j+jj)), 2));
                    //weighting gaussian
                    float weight = (float)(dist*(1/(distMin*paramA))+1)/(exp(dist
                        *(1/(distMin*paramA))));
                    if(mtSeg_tab[d][i+ii][j+jj]) pixelWhite += weight;
                    else pixelBlack += weight;
                }
            }

        if(pixelWhite>pixelBlack) mt_out[i][j]=255;
        else mt_out[i][j]=0;
    }
```

```
}
```

7.6 Algorithme de transformation log-polaire et log-polaire inverse

Issu de [6].

```
//-------------------------------------------------------------
// module   : FunctionBIEDA.cc
// auteur   : Mignotte Max
// labo     : DIRO
//-------------------------------------------------------------

void LogPolarMapping(float*** imgIn,float*** imgOut,int lgth,int wdth,int RowDec,int
     ColDec,int inverse,int BilInterp)
{
    int i,j,k;
    int col_inf,col_sup;
    int row_inf,row_sup;
    float row,col;
    float t,u;
    float ng;

    float Dist,Angle;
    const float RADIAN=2.0*PI/360.0;

    int RowCenter=(int)((lgth/2))+RowDec;
    int ColCenter=(int)((wdth/2))+ColDec;
    double WDTH_MAX=360.0;
    double LGTH_MAX=log(sqrt(CARRE((lgth/2)+abs(RowDec))+CARRE((wdth/2)+abs(ColDec))
        ));
    double ScalWdth=WDTH_MAX/(double)wdth;
    double ScalLgth=LGTH_MAX/(double)lgth;

    //Init
    for(k=0;k<3;k++)
        for(i=0;i<lgth;i++) for(j=0;j<wdth;j++) imgOut[k][i][j]=0;

    //PolarMapping
    for(k=0;k<3;k++)
        for(i=0;i<lgth;i++) for(j=0;j<wdth;j++)
        {
```

```
            if (!inverse)
            { Dist=exp(((float)i*ScalLgth));
                row=RowCenter+(Dist*sin(j*ScalWdth*RADIAN));
37          col=ColCenter+(Dist*cos(j*ScalWdth*RADIAN));  }

            if (inverse)
            { Dist=log(sqrt(CARRE(i-RowCenter)+CARRE(j-ColCenter)));
                Dist/=ScalLgth;
42          Angle=atan2((i-RowCenter),(j-ColCenter));
                if (Angle<0) Angle+=2*PI;
                Angle*=wdth/((float)(2.0*PI));
                row=Dist;
                col=Angle; }
47
            if ((int)row>(lgth-1))  continue;
            if ((int)row<0)         continue;
            if ((int)col>(wdth-1))  continue;
            if ((int)col<0)         continue;
52
            col_inf=(int)(col);
            col_sup=(int)(col+1);
            row_inf=(int)(row);
            row_sup=(int)(row+1);
57
            if (col_sup>(wdth-1)) col_sup=(wdth-1);
            if (col_sup<0)        col_sup=0;
            if (col_inf>(wdth-1)) col_inf=(wdth-1);
            if (col_inf<0)        col_inf=0;
62          if (row_sup>(lgth-1)) row_sup=(lgth-1);
            if (row_sup<0)        row_sup=0;
            if (row_inf>(lgth-1)) row_inf=(lgth-1);
            if (row_inf<0)        row_inf=0;

67          //LinearInterpol
            imgOut[k][i][j]=imgIn[k][(int)row][(int)col];

            //BilinearInterpol
            if (BilInterp)
72          {
                t=u=0.0;
                if (col_sup-col_inf) t=(col-col_inf)/(col_sup-col_inf);
                if (row_sup-row_inf) u=(row-row_inf)/(row_sup-row_inf);
```

```
77                  ng=(1-t)*(1-u)*imgIn[k][row_inf][col_inf];
                    ng+=t*(1-u)*imgIn[k][row_inf][col_sup];
                    ng+=t*u*imgIn[k][row_sup][col_sup];
                    ng+=(1-t)*u*imgIn[k][row_sup][col_inf];
                    if (ng>255) ng=255;
82                  if (ng<0)   ng=0;
                    if isnan(ng) printf("*");
                    imgOut[k][i][j]=(int)ng;
                }
            }
87  }
```