INF4173 – PROJET SYNTHÈSE

RAPPORT FINAL

Détection de piétons dans la vidéo surveillance
en utilisant l'algorithme Adaboost.

Présenté à

M. Michal Iglewski
M. Mohand Saïd Allili

Par

Pablo PEDROCCA

Département d'informatique et d'ingénierie
Université du Québec en Outaouais
23 avril 2010

Table de Matières

Table des Figures

1 Sommaire

Le but du projet consiste à développer une approche de détection de piétons dans une séquence vidéo à l'aide de l'algorithme de classification Adaboost. L'algorithme Adaboost permet de construire une frontière non-linéaire entre deux classes de données, en combinant plusieurs classificateurs. Par ce fait, nous pourrons, dans une vidéo de surveillance, séparer aisément des objets ressemblant à des piétons d'autres objets détectés.

Ce rapport est divisé en cinq sections. D'abord, le Chapitre I introduit quelques concepts fondamentaux de traitement d'images avec Matlab (notre outil de développement). Ensuite, le Chapitre II aborde le problème de soustraction de fond et l'extraction des caractéristiques d'une silhouette pour obtenir les données nécessaires.

Le Chapitre III fait une description détaillée de l'algorithme Adaboost et son mode de fonctionnement. La façon dont l'algorithme Adaboost a été intégré à la soustraction de fond et l'extraction des caractéristiques est abordée dans le Chapitre IV, ainsi que les résultats obtenus, les problèmes rencontrés et l'effort fourni lors de l'implantation.

Finalement, le Chapitre V fait une conclusion du travail et énonce quelles seront des étapes qui pourraient être poursuivies pour améliorer les résultats

L'évolution du projet par rapport au plan de travail est très satisfaisante, étant donné que toutes les tâches ont été accomplies dans les délais prévus. Les difficultés trouvées ont été surmontées, bien qu'elles aient provoqué quelques délais. Finalement, il faut remarquer que nous avons achevé le but visé, c.-à-d. classifier (classer) correctement les piétons dans une séquence vidéo.

2 Chapitre I – Le traitement des images – notions préalables

2.1 Le traitement d'images avec MATLAB

MATLAB est un outil logiciel qui fournit une grande librairie de traitement d'images (parmi plusieurs autres). Les avantages d'une telle librairie est qu'on enlève du programmeur la tâche fastidieuse de traiter chaque image « à la main », et ainsi on cache la complexité du traitement.

L'inconvénient des librairies de MATLAB est qu'elles ne sont pas optimisées : Elles ont été conçues plutôt pour faire une preuve de concept, et le traitement n'est pas rapide. En conséquence, le présent travail servira plutôt comme preuve de concept, tandis qu'une implantation commerciale devra être portée vers un autre langage (Java ou C++)

MATLAB nous offre plusieurs catégories de fonctions pour le traitement d'images. On fera le sommaire des fonctions qu'on utilise dans notre projet.

Catégorie	**Fonction**	**Description**
Lecture d'images d'une source vidéo	Avi_size	Permet d'extraire les caractéristiques d'un fichier .AVI, tel que taille de l'image et nombre de cadres
	aviread	Permet de lire une image particulière d'un fichier vidéo
	Frame2im	Permet de convertir l'image lue avec aviread à une image RGB
Manipulation des images	Im2double	Permet de convertir une image dont chaque pixel est représenté par un octet à une image dont chaque pixel est représenté par un nombre du type double
	imabsdiff	Fait la valeur absolue de la soustraction de deux images A et B, ou chaque pixel est donnée par |A-B|
	Rgb2gray	Fait la conversion d'une image RGB vers une image en échelle de grises (à niveau de gris)
	Imfill	Cette fonction est utilisée pour remplir des trous dans une image binaire
	Imfilter	Avec cette fonction on applique des différents types de filtres à l'image pour, par exemple, éliminer le bruit.
Transformations Morphologiques	Imdilate	Applique une opération de dilation sur l'image
	Imerode	Applique une opération d'érosion sur l'image
Général	Imshow	Montre une image à l'écran

2.2 Quelques algorithmes de traitement d'images

Il y a plusieurs algorithmes de traitement d'images, et chacun répond à un problème particulier. Il y a un nombre limité d'algorithmes que nous allons utiliser dans notre projet, et on peut les classer dans les catégories suivantes :

2.2.1 Algorithmes pour améliorer la qualité des images

Ces algorithmes fonctionnent en appliquant des filtres aux images. Un filtre très commun est le *filtre médian*, aussi connue comme « sel et poivre », car il permet d'éliminer d'une image le bruit qui ressemble au sel et poivre. Ce filtre parcourt une image M et établit pour chaque pixel de sortie une valeur définie par la médiane de tous les pixels voisins.

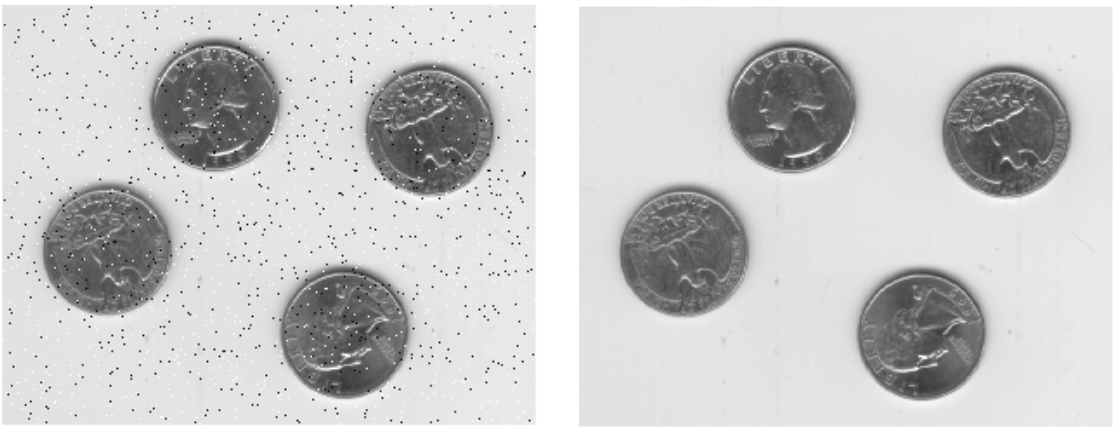


Figure 1 a) Image avec bruit « sel et poivre » b) Image après l'application d'un filtre médiane.

La fonction qu'on utilise en MATLAB est medfilt2, et le paramètre à établir est le nombre d'éléments dans le voisinage, par exemple 3x3; ça veut dire que la valeur d'un pixel P sera donnée par la médiane de son voisinage de 3x3 pixels.

2.2.2 Algorithmes pour les opérations morphologiques

Dans notre projet, nous utilisons deux opérations morphologiques : la dilation et l'érosion. Ces opérations nous permettent de changer la forme d'un objet dans l'image (toujours par rapport à sa forme originale) pour mettre en évidence certaines caractéristiques.

Pour le dire d'une façon plutôt simpliste, l'opération de dilation est équivalente à « gonfler » un objet, tandis qu'une opération d'érosion est l'équivalente à le « dégonfler ».

Dans une opération de dilation, la valeur de chaque pixel de sortie est la valeur maximale de tous les pixels de son voisinage. Il y a plusieurs façons de définir un tel voisinage. Pour l'instant, on se contente d'établir un voisinage carré de 3x3.

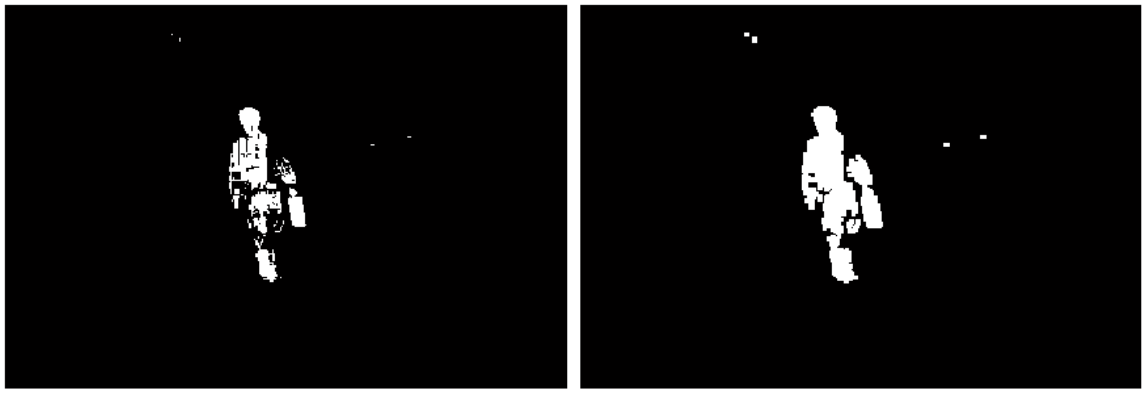


Figure 2 a) Image originale b) Image après dilation

On peut voir ci-dessus une image avant et après une opération de dilation. On remarque que les « trous » dans l'image ont été remplis et l'image a un air plus homogène.

Dans une opération d'érosion, la valeur de chaque pixel de sortie est la valeur minimale de tous les pixels de son voisinage. Cela a comme résultat un objet un peu plus « maigre » que l'objet original.

Figure 3 a) Image originale b) Image après érosion

On peut voir ci-dessus une image avant et après une opération d'érosion. On remarque que l'objet a un air un peu plus homogène que celle de la figure antérieure.

3 Chapitre II – La soustraction de fond et l'extraction des caractéristiques

3.1 Algorithmes de soustraction de fond

Le domaine de soustraction de fond (« background substraction (subtraction) ») est soi-même immense et a été le sujet de plusieurs recherches. On peut définir la soustraction de fond comme étant l'opération qui fixe une image de fond, autre image comme front, et fait une série d'opérations mathématiques pour obtenir les différences entre elles.

Dans le cadre de notre projet on a essayé plusieurs algorithmes, et on montrera les résultats obtenus avec chacun. Pour nos tests, on a utilisé les images suivantes :

Figure 4 Images d'essai a) Le background b) Image avec un piéton.

3.1.1 La distance euclidienne

Cet algorithme calcule la valeur absolue de la différence entre chaque pixel. Alors
M' = | M2 – M1 |

3.1.2 Différence par division

Cet algorithme calcule chaque pixel de la sortie comme la différence entre chaque pixel de l'image du front avec chaque pixel du background. Alors
M' = M2 / M1

3.1.3 Combinaison de la distance euclidienne et la division

Ici, on combine les deux algorithmes mentionnés ci-haut. Alors chaque image résultante est calculée comme suit :
M1' = | M2 – M1 |
M2' = M2 / M1
M' = M1' + M2'

Les résultats de ces méthodes sont montrés ci-dessous :

Figure 5 a) Algorithme de différence b) Algorithme de division c) Combinaison des algorithmes.

En tenant compte de la simplicité versus la qualité des résultats, l'algorithme de soustraction par distance euclidienne a été choisi pour notre projet.

3.2 Extraction des caractéristiques géométriques

Une fois la soustraction du fond faite, on peut obtenir un « blob » qui représente l'objet qui nous intéresse. Alors, on procède à extraire des caractéristiques qui nous permettront plus tard de définir si l'image appartient ou non à un humaine.

Pour définir les parties composantes de la silhouette, on divise le « blob » de la façon suivante :

Partie	**Taille**	**Fonction**
Tête	¼ de la taille totale	Section pour la tête
Torse	1/3 de la taille totale	Section pour le torse
Jambes	2/3 de la taille totale	Section pour les jambes

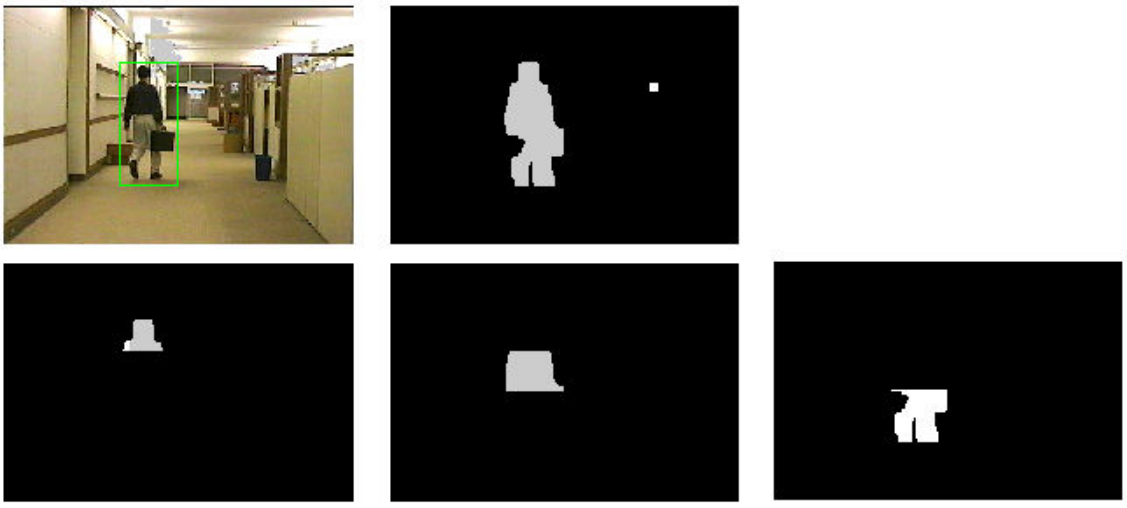


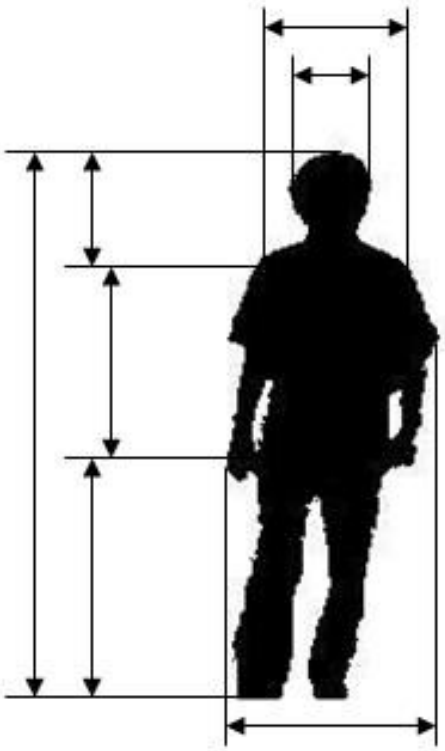
Figure 6 a) Image originale b) blob extrait après la soustraction de background et le seuillage (threesholding) c, d, e) les sous-sections de la tête, le torse et les jambes respectivement.

La prise des mesures à intervalles discrets nous permettra d'extraire des caractéristiques. Par exemple, une prise de la mesure de la taille mineure versus la taille majeure de la tête pourra nous donner une relation tête/épaules qui est soi-même une caractéristique.

Figure 7 Points de mesure pour établir la relation tête/épaules

Dans ce projet, nous avons identifié les caractéristiques suivantes:

- Ratio hauteur/largeur de la silhouette.
- Ratio tête/épaules
- Ratio épaules/tronc
- Ratio surface tête/surface silhouette.
- Ratio surface tronc/surface silhouette.
- Ratio surface jambes/surface silhouette.



Les caractéristiques mentionnées ci-dessus seront extraites par l'algorithme d'extraction des caractéristiques et fournies à l'algorithme de classification Adaboost.

4 Chapitre III - L'algorithme Adaboost

L'algorithme Adaboost est un algorithme de « amplification adaptative » (*adaptative boosting*, d'où son nom). Cet algorithme permet de construire une frontière non linéaire entre deux classes de données; c'est-à-dire, il est un algorithme de classification binaire. Dans notre application, il nous permettra de séparer les objets entre « piétons » et « non-piétons ».

Adaboost est un algorithme intelligent qui a une capacité d'apprentissage: on lui fournit un ensemble des données positives et négatives avec une étiquette pour chaque exemple indiquant sa classe; Adaboost tracera alors une frontière non-linéaire entre les deux ensembles de données et gardera le modèle non-linéaire résultant. Au fur et à mesure que les nouvelles données arrivent, Adaboost fera une comparaison avec son modèle non linéaire existant et fera la classification des données.

Adaboost repose sur la sélection itérative de classificateurs faibles en fonction d'une distribution des exemples d'apprentissage. Chaque exemple est pondéré en fonction de sa difficulté avec le classificateur courant. Adaboost fait appel de façon répétée à un classificateur faible dans une série d'itérations t=1,…, T (T est définie en bas). Pour chaque appel une distribution des poids D_t est mise à jour; cette distribution D_t donne l'importance des exemples dans chaque ensemble de données. Dans chaque itération, le poids relatif de chaque donnée incorrectement classée est augmenté, de façon que chaque nouveau classificateur se concentre plus sur ces données.

Alors, le modèle non-linéaire où **classificateur « fort** » est composé de la **combinaison linéaire** de **classificateurs faibles**:

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

Où H(x) est le classificateur « fort », et il a deux valeurs possibles : +1 ou -1 (donné par la fonction ***sign***). H(x) est la sommation des classificateurs simples pondérés. Alors :

- La valeur T est le nombre de classificateurs faibles qu'on utilisera pour trouver le classificateur fort. Il est évident que si T est grand, l'efficacité du classificateur H(x) sera plus précise. Cependant, chaque itération en T prend du temps de traitement. Alors il nous faut trouver une valeur de T qui soit assez petite pour garantir un temps de traitement raisonnable et au même temps, assez grand pour que le classificateur H(x) soit efficace
- La fonction $h_t(x)$ est un classificateur « **faible** » et « **simple** ». Par exemple, une fonction pourrait être l'équation d'une ligne droite. C'est un classificateur linéaire qui est capable de diviser l'ensemble des données en deux.
- La valeur α_t est un nombre réel qui donne la pondération du classificateur $h_t(x)$, et qui est souvent calculé comme suit :

$$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$$

où ε_t est l'erreur pondéré du classificateur $h_t(x)$.

Fonctionnement de l'algorithme :

Soit une série de données $(x_1,y_1),\ldots,(x_m,y_m)$ ou $x_i \in X$, $Y_i \in \{-1,+1\}$. X est l'ensemble des données, Y sont les étiquettes de ces données (positifs ou négatifs).

1. Initialiser une distribution $D_1(i) = 1/m$, pour i=1,…,m; c'est-à-dire, chaque donnée a le même poids dans la distribution.

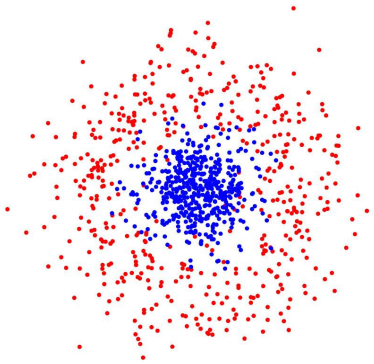


Figure 8 – Distribution des données

2. On fait appel T fois à un classificateur faible. Pour t = 1,…,T:

a. Trouver le classificateur faible ht qui minimise l'erreur par rapport à la distribution D_t:

$$h_t = \arg\min_{h_j \in \mathcal{H}} \epsilon_j \quad \text{où} \quad \epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

Si ε_t >=0.5 alors arrêt.

On avait dit que le classificateur faible h_t pourrait être, par exemple, l'équation d'une droite, alors l'ensemble de données aura l'air comme suit:

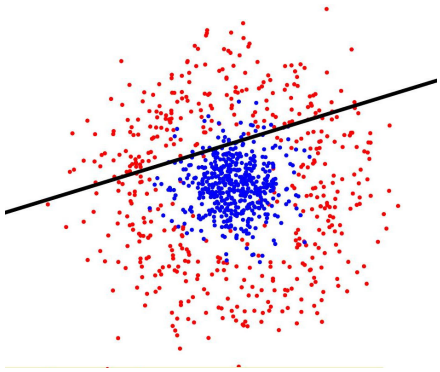


Figure 9 – Le classificateur faible

b. Choisir α_t tel que : $$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$$

où ε_t est l'erreur pondéré du classificateur $h_t(x)$.

c. Mettre à jour le paramètre de distribution:

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t \cdot y_i \cdot h_t(x_i))}{Z_t}$$

Où Z_t est un factor de normalisation (choisi d'une façon telle que D_t sera une distribution des probabilités).

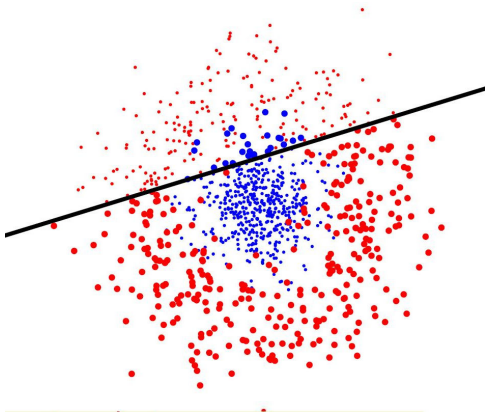


Figure 10 – Distribution corrigée

Ce qui montre que les poids relatifs des données qui ne sont pas bien classifiées seront plus importants lors de l'itération t+1.

Exemple de succession des itérations T (classificateurs faibles $h_t(x)$) :

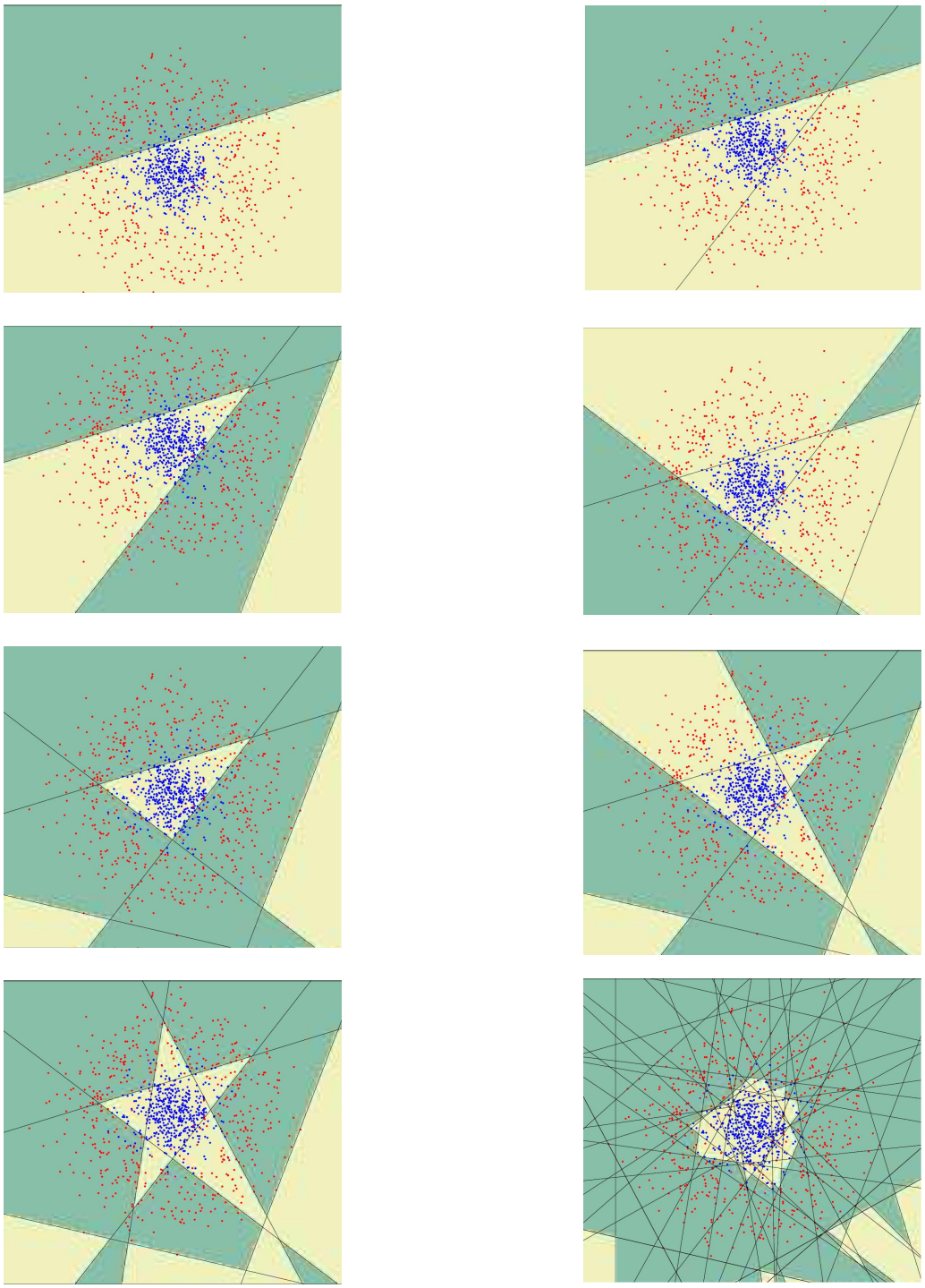


Figure 11 – Exemples des classificateurs faibles

5 Chapitre IV - Implantation

Dans le cadre de ce projet on n'a fait qu'établir une preuve de concept, c.-à-dire, la preuve que la classification de piétons en utilisant l'algorithme Adaboost est une avenue de recherche prometteuse. Par conséquent, on n'a pas travaillé sur l'optimisation du système en termes de vitesse ou interface utilisateur.

Comme il a été mentionné dans les sections précédentes, l'implantation de l'algorithme est faite avec MATLAB V.7. MATLAB est un outil logiciel qui fournit une grande librairie de traitement d'images (parmi plusieurs autres), ainsi qu'un nombre important de fonctions mathématiques.

5.1 Schéma globale du système

Il y a deux systèmes en un : d'abord, nous avons le système d'apprentissage supervisé, qui prend une base de données des silhouettes, extrait des caractéristiques géométriques, et génère le modèle non linéaire à l'aide de l'algorithme Adaboost-training. Une fois le modèle non linéaire généré, on peut prendre des séquences vidéo réelles et extraire les silhouettes, détecter les caractéristiques, et évaluer ces caractéristiques contre le modèle non linéaire à l'aide de l'algorithme Adaboost-évaluation.

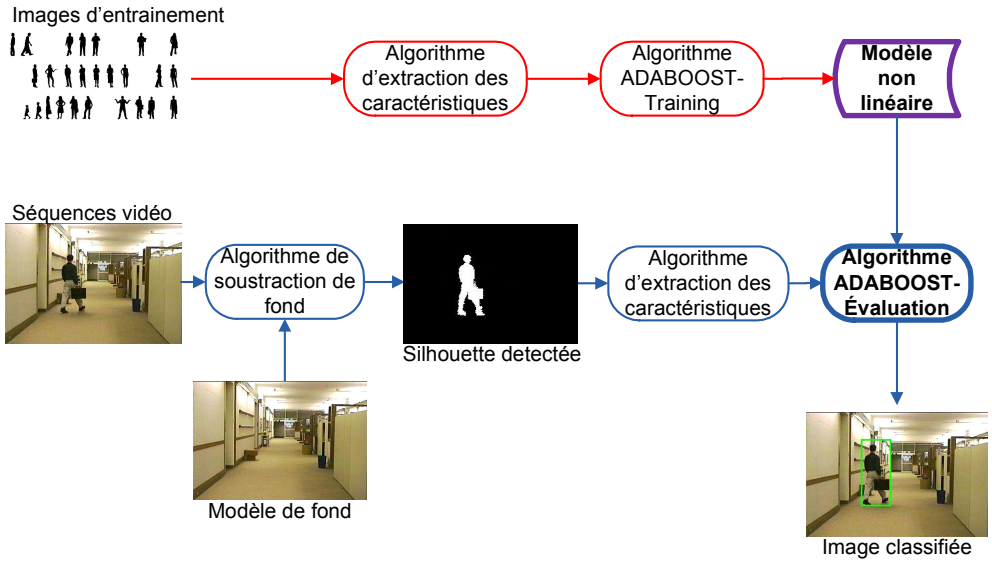


Figure 12 – Schéma globale du système

5.2 Le processus d'entrainement

Pour le processus d'entrainement on n'utilise que 20 classificateurs faibles (T=20); ce nombre de classificateurs a été trouvé d'une façon empirique et semble être un bon compromis entre la précision de la classification et le temps d'exécution.

5.2.1 Les données d'entrainement

Pour le processus d'entrainement d'Adaboost, nous avons utilisé 139 silhouettes synthétiques de très bonne qualité, étant 51 positifs (correspondant à des piétons) et 88 négatives. La source des données est un ensemble d'images statiques en format JPG; chaque fichier JPG peut contenir plusieurs silhouettes.

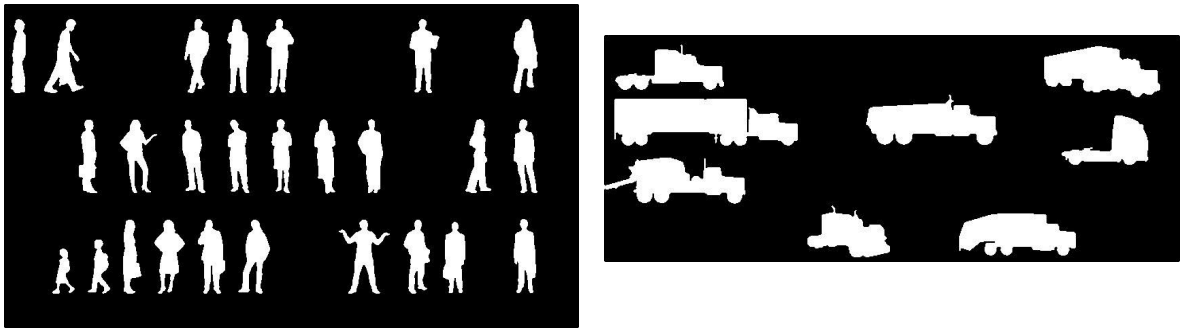


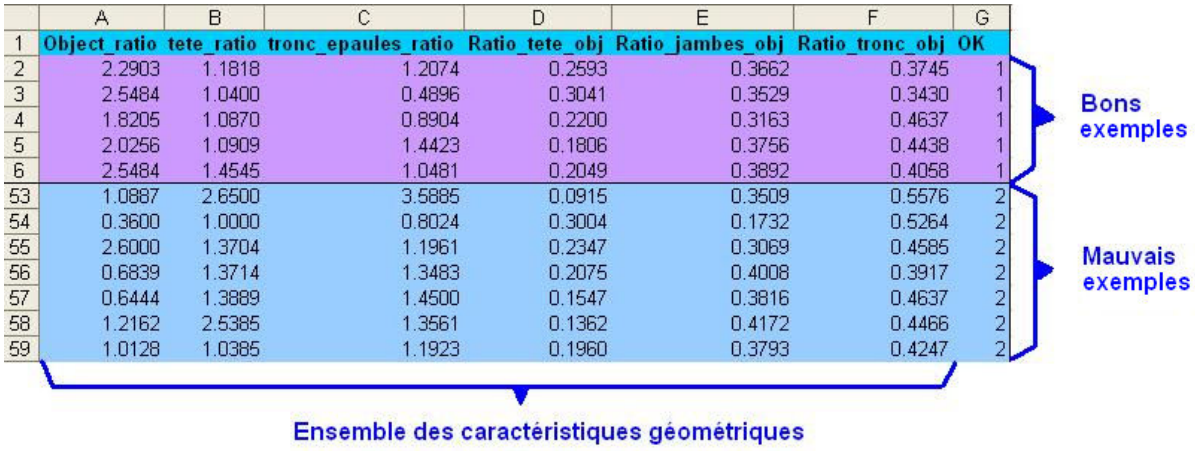
Figure 13 – Silhouettes d'exemple

L'algorithme de génération de la base de données des caractéristiques (**GenererEnsembleExample.m**) fera le parcours de chaque fichier, et individualisera chaque silhouette à l'aide de la fonction MATLAB ***bwboundaries(A)***, où A est l'image en traitement. Cette fonction nous retourne un tableau avec les différentes blobs dans l'image.

Une fois chaque blob individualisé, on fait l'extraction des caractéristiques en appelant la fonction ***ExtraireCaracteristiques*** de la façon suivante :

```
[tr_object_ratio, tr_tete_ratio, tr_epaules_ratio, ...
tr_tete_corps_ratio , tr_jambes_corps_ratio , ...
 tr_tronc_corps_ratio ] = Extraire_Caracteristiques(A, bnd);
```

Où A est l'image globale et bnd est une structure avec les éléments du blob. La fonction retourne les caractéristiques qui appartiennent au blob. Toutes ces caractéristiques pour cette blob seront écrites comme une rangée dans un fichier Excel. Une fois toutes les silhouettes traitées, le fichier Excel aura l'aspect suivant :



	A	B	C	D	E	F	G
1	Object_ratio	tete_ratio	tronc_epaules_ratio	Ratio_tete_obj	Ratio_jambes_obj	Ratio_tronc_obj	OK
2	2.2903	1.1818	1.2074	0.2593	0.3662	0.3745	1
3	2.5484	1.0400	0.4896	0.3041	0.3529	0.3430	1
4	1.8205	1.0870	0.8904	0.2200	0.3163	0.4637	1
5	2.0256	1.0909	1.4423	0.1806	0.3756	0.4438	1
6	2.5484	1.4545	1.0481	0.2049	0.3892	0.4058	1
53	1.0887	2.6500	3.5885	0.0915	0.3509	0.5576	2
54	0.3600	1.0000	0.8024	0.3004	0.1732	0.5264	2
55	2.6000	1.3704	1.1961	0.2347	0.3069	0.4585	2
56	0.6839	1.3714	1.3483	0.2075	0.4008	0.3917	2
57	0.6444	1.3889	1.4500	0.1547	0.3816	0.4637	2
58	1.2162	2.5385	1.3561	0.1362	0.4172	0.4466	2
59	1.0128	1.0385	1.1923	0.1960	0.3793	0.4247	2

Figure 14 – Données d'entrainement

5.2.2 La génération du modèle non linéaire

Une fois que les caractéristiques de toutes les silhouettes (positives et négatives) sont dans la base de données (fichier Excel) on applique un algorithme de génération du modèle non linéaire (**Init_Training_Set.m**)

D'abord, toutes les N rangées du fichier Excel sont mises dans un tableau matData [N, 6]. On traite seulement les 6 premières colonnes (celles qu'on les caractéristiques). La septième colonne représente les étiquettes et elle est mise dans le tableau tr_labels.

Pour obtenir le modèle non linéaire, on appelle l'algorithme Adaboost-training de la façon suivante :

```
% Weak_learner est la quantité de classificateurs faibles à utiliser
% (Notre valeur T)
weak_learner_n = 20;

% On appelle Adaboost T fois
for i=1:weak_learner_n
      adaboost_model = ADABOOST_tr(@threshold_tr, @threshold_te,
                                                   matData,tr_labels,i);

end
```

Ce processus de génération du modèle non linéaire pourrait être exécuté chaque fois que l'algorithme d'évaluation démarre. Par souci de rapidité, on a décidé de garder la variable ***adaboost_model*** dans un fichier, et de la charger plus tard.

5.3 L'évaluation des séquences vidéo

5.3.1 La soustraction de fond

Il faut remarquer que la soustraction de fond ne fait pas partie du cadre de ce projet. Elle est implémentée seulement car on avait besoin d'elle pour extraire les silhouettes des séquences vidéo.

Une fois la séquence vidéo ouverte en utilisant les fonctions ***aviread*** de MATLAB, on prend la première image et on la sauve en mémoire; cette image sera notre modèle de fond (background model).

À chaque image lue de la séquence vidéo, on appelle la fonction ***Soustraire_Background*** comme suit :

```
%lecture du cadre actuel
cadreActuel = aviread(m_fichierVideo,cadres);
%extraction de l'image courrante
[frame,MapFrame] = frame2im(cadreActuel);
%on affiche l'image
subplot(2,2,1), imshow(frame, MapFrame);

%Conversion de l'image à double
cadre_actuel = im2double(frame);
%A = image avec les silhouettes (Y est le modèle de fond).
A = Soustraire_Background(cadre_actuel, Y, X, cadre_anterieur);
```

5.3.2 La classification des silhouettes

Une fois que l'image A est obtenue en utilisant la soustraction de fond, on extrait les blobs de l'image et les caractéristiques de la même façon qu'on l'a fait dans l'algorithme d'entrainement :

```
[tr_object_ratio, tr_tete_ratio, tr_epaules_ratio, ...
tr_tete_corps_ratio , tr_jambes_corps_ratio , ...
 tr_tronc_corps_ratio ] = Extraire_Caracteristiques(A, bnd);
```

Mais cette fois-ci, les variables ainsi obtenues sont mises dans un tableau de dimensions [1 6] qui ne contient que les caractéristiques. Un deuxième tableau pour les étiquettes de dimensions [1 1] est mise à la valeur 1, c.-à-d. l'objet par défaut est humain (car on préfère avoir un faux positif). Il faut remarquer que dans cette implantation de l'algorithme on utilise la valeur 1 pour les étiquettes positives, et 2 pour les étiquettes négatives.

Avec ces valeurs, on appelle l'algorithme Adaboost-évaluation. On envoie comme paramètres les variables ***adaboost_model*** (le modèle non linéaire obtenue dans la phase d'entrainement), ***te_set*** (les caractéristiques) et ***te_labels*** (les étiquettes) :

```
% Étiquette
te_labels(1,1) = 1;
% Caractéristiques
te_set(1,1) = tr_object_ratio;
te_set(1,2) = tr_tete_ratio;
te_set(1,3) = tr_epaules_ratio;
te_set(1,4) = tr_tete_corps_ratio;
te_set(1,5) = tr_jambes_corps_ratio;
te_set(1,6) = tr_tronc_corps_ratio;

% Appel à Adaboost_te (Adaboost TEST, ou Évaluation)
 [L_te,hits_te] = ADABOOST_te(adaboost_model, @threshold_te,
                                          te_set,te_labels);
```

Ici on n'a qu'une seule rangée, marquée avec une étiquette 1 (piéton). Alors, si les caractéristiques correspondent effectivement à un piéton, la variable ***hits_te*** retournera la valeur 1, 0 sinon.

Si la valeur de ***hits_te*** est 1, on dessine un rectangle vert autour du blob, sinon on dessine un rectangle rouge.

5.3.3 Résultats obtenues et problèmes rencontrés

L'implantation d'Adaboost fut testée avec deux types des silhouettes : des silhouettes statiques et des séquences vidéo. Le test avec des silhouettes statiques a été fait pour prouver les capacités de l'algorithme pour identifier les silhouettes sans les inconvénients provoqués par la soustraction de fond dans les séquences vidéo.

En général, la classification avec des silhouettes statiques a donné des bons résultats. On peut remarquer que l'algorithme n'est pas capable de catégoriser une silhouette comme humaine si elle est attachée à un autre objet (exemples : le monsieur encombré sur la voiture ou l'ouvrier qui porte la charrette) :

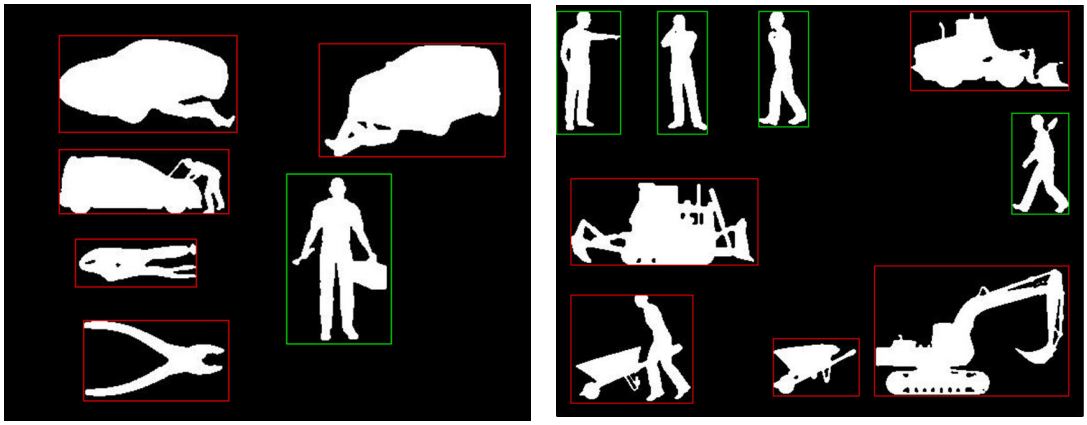


Figure 15 – Résultats obtenues dans des images statiques

En ce qui concerne les séquences vidéo, l'algorithme était capable de classifier les silhouettes avec les mêmes restrictions que dans les silhouettes statiques. Il est pourtant plus faible à cause du bruit et de l'interférence générée par la soustraction de fond.

Figure 16 – Résultats obtenues dans des séquences vidéo

Pendant l'exécution de ce projet on a trouvé plusieurs difficultés. Parmi eux, la soustraction de fond et la détermination des caractéristiques demeurent les principaux items à résoudre.

Bien qu'elle ne fasse pas partie de ce projet, la soustraction de fond est nécessaire pour achever la détection et classifications des blobs dans des séquences vidéo. Or, un bon algorithme de soustraction de fond donne des silhouettes plus distinctives. Les algorithmes essayés ont toujours beaucoup d'imperfections, laissant les silhouettes avec « trous » ou carrément incomplètes. Les algorithmes essayés détectent aussi les ombres projetées par les piétons comme faisant partie des blobs, et ils ne sont pas sensibles aux changements du fond.

La détermination des caractéristiques a été faite d'une façon plutôt empirique et expérimentale, n'ayant pas des expériences antérieures dans ce domaine. Alors, on a trouvé des caractéristiques semblant les bonnes, mais on peut travailler là-dessus pour trouver d'autres plus distinctives.

5.3.4 Effort fourni lors de l'implantation

Pendant tout ce projet il a fallu entreprendre les activités suivantes :

- Lecture et analyse de publications reliées à ce projet.
  Plusieurs papiers scientifiques ont été lus pour trouver des idées et mieux comprendre la problématique à résoudre.

- Acquisition des connaissances de traitement d'images avec MATLAB
  La documentation de MATLAB a été lue et on a acquis une connaissance des différentes fonctions de MATLAB pour le traitement d'images.

- Étude de quelques algorithmes de traitement d'images (égalisation d'histogrammes, réduction de bruit, soustraction de fond).
  Ces algorithmes ont été étudiés et implémentés postérieurement pour améliorer les images résultantes de la soustraction de fond.

- Acquisition des notions théoriques sur la classification Adaboost.
  Cette tâche est toujours en développement; bien que l'algorithme soit compliqué, son implantation est simple, mais on essaye de bien comprendre son fonctionnement avant de procéder à l'implantation

- Soustraction de fond avec plusieurs algorithmes.
  Plusieurs algorithmes de soustraction on été essayés; quelques uns étaient simples et basés sur la distance euclidienne entre deux vecteurs; les autres étaient un peu plus complexes et travaillaient sur les composants de la couleur de chaque image. Finalement, en vue de la simplicité de l'implantation et les résultats, on a décidé de rester avec l'algorithme le plus simple, tout en

appliquant des algorithmes a posteriori pour améliorer le résultat. On a réussi à individualiser les différentes blobs dans l'image.

- Extraction des caractéristiques
  Une fois les blobs extraits, on estime la relation hauteur/largeur du blob, pour voir si on présélectionne le blob comme étant un piéton. Si oui, on divise chaque blob en trois parties, pour les jambes, le torse et la tête. Une fois cela fait, ce qui reste est de prendre des mesures relatives de chaque partie, ce qui donneront nos caractéristiques.

- Implantation d'Adaboost.
  On a pris une implantation existante de l'algorithme Adaboost, écrite en Matlab par Cuneyt Mertayak (voir références). Il a fallu analyser et bien le comprendre pour procéder à son utilisation.

- Application d'Adaboost pour la détection des piétons.
  Après l'extraction de caractéristiques, l'algorithme Adaboost a été appliqué pour la détection des piétons.

- Validation de vidéos réelles.
  Cette étape inclut des tests validité de l'algorithme qui furent exécutés sur des vidéos réelles et sur des images statiques.

- Présentation du projet
  Présentation du projet devant jury, exposant le projet lui-même, les résultats, et avec une période des questions et réponses.

- Remise du rapport final.
  Écriture de ce rapport, étant le livrable ultime du projet, il présente le détail du travail réalisé et les résultats achevés.

6 Chapitre V – Conclusion, prochaines étapes, et améliorations à apporter

On parle ici d'une avenue de **recherche** prometteuse, car il y a beaucoup d'applications. Cela n'est pas restreint seulement à la surveillance : on pourrait implémenter le système de détection des piétons dans, par exemple, un système d'assistance aux conducteurs des voitures. D'autre part, étant donné qu'il s'agit d'un système de classification binaire entrainé avec des caractéristiques, on pourrait l'utiliser dans une ferme pour faire la distinction entre mouton / non-mouton, par exemple.

Aussi, j'ai développé un intérêt à continuer le développement pour en surmonter les difficultés rencontrées et améliorer le système. Parmi les améliorations à apporter, on peut:

- Implémenter un algorithme de soustraction de fond plus robuste
- Trouver d'autres caractéristiques plus discriminantes
- Extraire des caractéristiques de l'intérieur de la silhouette (ex. est-ce que la silhouette a un visage? Si oui, est-ce qu'il est bien formé?).
- Détecter les objets à travers des occlusions.
- Détecter partiellement des objets (ex. piéton partiellement caché).
- Détecter des groupes d'humains.

En conclusion, ce fut un projet fort intéressant en intelligence artificielle, qui permet de combiner le **traitement d'images** et **l'apprentissage statistique**. Bien que difficile, on a réussi à compléter tout le travail et achever les objectifs attendues.

7 Annexe I - Bibliographie

[1] Gonzalez, R.C. & Woods, R.E. (2007). Digital Image Processing. Upper Saddle River : Prentice Hall.

[2] Lin Z. & Davis, L.S. (2009). Shape-based Human Detection and Segmentation via Hierarchical Part-Template Matching. IEEE Transactions on pattern analysis and Machine Intelligence

[3] Kyungnam Kim, Thanarat H. Chalidabhongse, David Harwood, Larry Davis (2005) Real-time foreground–background segmentation using codebook model . www. Sciencedirect.com

[4] Stephen J. McKenna, Sumer Jabri, Zoran Duric, Azriel Rosenfeld and Harry Wechsler (2000).Tracking Groups of People. Computer Vision and Image Understanding **80,** 42–56 (2000)

[5] Cuneyt Mertayak, Implèmentation de l'algorithme Adaboost avec Matlab, (http://www.mathworks.com/matlabcentral/fileexchange/authors/31615

8 Annexe II : Code Matlab

8.1 Programme d'analyse des images statiques

```
function debut_statique()

%Cette programme debut fait l'analyse des images statiques
%on a deja une silhouette
clear all
%initialize le modele d'adaboost et la base des données
path_donnees = 'C:\Pablo\University\Hiver 2010\Projet
Synthese\Source\Adaboost\';
fichier_donnees = 'training_set.xls';
source_donnees = strcat(path_donnees, fichier_donnees);
%adaboost_model = Init_Training_Set (source_donnees);
%save(strcat(path_donnees,'adaboost_model.inf'),'adaboost_model');
m_fichier_model = strcat(path_donnees,'adaboost_model.inf') ;
load (m_fichier_model, '-mat', 'adaboost_model');

aviobj = avifile(strcat(path_donnees,'test1.avi'),
'Colormap',gray(256));
aviobj.quality = 100;
aviobj.fps=1;
aviobj.compression = 'None';

fig=figure;
set(fig,'DoubleBuffer','on');
set(gca,'xlim',[-80 80],'ylim',[-80 80],...
    'nextplot','replace','Visible','off')

%maintenant, on prend les images de test.
m_pathsource = 'C:\Pablo\University\Hiver 2010\Projet
Synthese\Images\Test\';
linefile = 0;
mymatrix = [];
files = dir(m_pathsource )
numfiles = size(files,1)
fp=3;
while (fp<numfiles+1)
    if (files(fp).isdir)
        fp = fp+1;
        continue
    end
    m_Image = files(fp).name;
    m_fichierImage = strcat(m_pathsource,m_Image);

    A = imread(m_fichierImage);

    subplot(1,1,1),    imshow(A);


       % on calcule les limites de tous les blobs dans l'image
```

```
    boundaries = bwboundaries(A);
    bndrSize = size(boundaries);
    for k = 1:bndrSize(1)
      bnd = boundaries{k};
      RB2 = max(bnd);
      LT2 = min(bnd);
      if ( (max(RB2-LT2)>20) && notInsideOther(boundaries,k) )

        %extraire les caracteristiques
        [tr_object_ratio, tr_tete_ratio, tr_epaules_ratio , ...
         tr_tete_corps_ratio , tr_jambes_corps_ratio ,
tr_tronc_corps_ratio ] = ...
            Extraire_Caracteristiques(A, bnd);

        %Est-ce un humain? Appeler adaboost pour en savoir.
        te_labels(5,1) = 2;
        te_set(5,1) = 0;
        te_labels(:,1)=2;
        te_labels(1,1) = 1;

        te_set(1,1) = tr_object_ratio;
        te_set(1,2) = tr_tete_ratio;
        te_set(1,3) = tr_epaules_ratio;
        te_set(1,4) = tr_tete_corps_ratio;
        te_set(1,5) = tr_jambes_corps_ratio;
        te_set(1,6) = tr_tronc_corps_ratio;

        [L_te,hits_te] =
ADABOOST_te(adaboost_model,@threshold_te,te_set,te_labels);
        %Dessiner les carres autour des objets
        object_width = abs(RB2(2)-LT2(2));
        object_height = abs(RB2(1)-LT2(1));
%        hits_te = 1;
        hold on;
        if ((object_width~=0) && (object_height~=0))
            if (hits_te==5)

rectangle('Position',[LT2(2),LT2(1),object_width,object_height],'EdgeCo
lor','g','LineWidth',1);
            else

rectangle('Position',[LT2(2),LT2(1),object_width,object_height],'EdgeCo
lor','r','LineWidth',1);
            end
        end
        drawnow;
        aviframe = getframe(gca);
        aviobj = addframe(aviobj, aviframe);
      end
    end
    fp = fp+1;
end
aviobj = close(aviobj);

function m_inside = notInsideOther(m_boundaries, m_currelem)
```

```
    m_inside=true;
    bndrSize = size(m_boundaries);
    curbnd = m_boundaries{m_currelem};
    curRB2 = max(curbnd);
    curLT2 = min(curbnd);
    f = 0;
    % maintenant, on parcours tous les blobs qu'on a dans l'image
    for k = 1:bndrSize(1)
        if (m_currelem~=k)
          bnd = m_boundaries{k};
          RB2 = max(bnd);
          LT2 = min(bnd);
          if (  (   (curLT2(2)>LT2(2) && curLT2(2)<RB2(2)) && ...
                    (curLT2(1)>LT2(1) && curLT2(1)<RB2(1)) ) || ...
                (   (curRB2(2)>LT2(2) && curRB2(2)<RB2(2)) && ...
                (curRB2(1)>LT2(1) && curRB2(1)<RB2(1))  )   )
                m_inside=false;
                return
          end
        end
    end
```

8.2 Programme d'analyse des séquences vidéo

```
function debut_dynamique_red_chair()

%Cette programme debut fait l'analyse des sequences video
clear all
%initialize le modele d'adaboost et la base des données
path_donnees = 'C:\Pablo\University\Hiver 2010\Projet
Synthese\Source\Adaboost\';
m_pathsource = 'C:\Pablo\University\Hiver 2010\Projet
Synthese\Images\Test\Video\';
fichier_donnees = 'training_set.xls';
source_donnees = strcat(path_donnees, fichier_donnees);
%adaboost_model = Init_Training_Set (source_donnees);
%save(strcat(path_donnees,'adaboost_model.inf'),'adaboost_model');
m_fichier_model = strcat(path_donnees,'adaboost_model.inf') ;
load (m_fichier_model, '-mat', 'adaboost_model');

aviobj = avifile(strcat(path_donnees,'testdynamique_red_chair2.avi'),
'Colormap',gray(256));
aviobj.quality = 100;
aviobj.fps=12;
aviobj.compression = 'None';
%scrsz = get(0,'ScreenSize');
%fig = figure('Position',[1 scrsz(4)/2 scrsz(3)/2
%scrsz(4)/2]);set(0,'Units','pixels') % Sets units to pixels
fullscreen = get(0,'ScreenSize');
fig=figure('Units','normalized','Position',[0 0 1 1]);
%p = get(0,'monitorpositions');
%set(gcf,'position',p(2,:)) ;

%set(fig, 'Position', [0 0 fullscreen(3) fullscreen(4) ] );
```

```
set(fig,'DoubleBuffer','on');
%set(gcf,'Units','normalized','Position',[-1.00 0.032 1024.00 2]);
set(gca,'xlim',[-160 160],'ylim',[-160 160],...
    'nextplot','replace','Visible','off')
winsize = get(fig,'Position');
    winsize(1:2) = [0 0];


m_fichierVideo = strcat(m_pathsource, 'RedChair.avi');
%m_fichierOrg = strcat(m_pathsource, 'hall_monitor.avi');
%m_fichierVideo = '..\..\images\office.avi';
[height, width, n_frames] = avi_size(m_fichierVideo);

videoBase = aviread(m_fichierVideo);

%maintenant, on prend les images de test.
linefile = 0;
mymatrix = [];
files = dir(m_pathsource )
numfiles = size(files,1)
fp=3;

cadreActuel = aviread(m_fichierVideo,1);
[frame,MapFrame] = frame2im(cadreActuel);


subplot(2,2,1),    imshow(frame);

Y = im2double(frame);
Y (Y==0) = 0.0001;
cadre_anterieur = Y;
misc = (Y(:,:,1)+Y(:,:,2)+Y(:,:,3));
misc(misc==0) = 0.0001;

X = Y;
%After aggregating R, G and B, it is checked that if any value is 0
then it is set 0.001 to overcome “divide by zero error”. Normalizing R
factor.
X(:,:,1) = X(:,:,1)./misc;
%Normalizing G factor
X(:,:,2) = X(:,:,2)./misc;

for cadres = 50:200  %n_frames-5
    %lecture du cadre actuel
%    frame = frame.*255;

    cadreActuel = aviread(m_fichierVideo,cadres);
    [frame,MapFrame] = frame2im(cadreActuel);
    subplot(2,2,1), imshow(frame, MapFrame);
    subplot(2,2,3), imshow(frame, MapFrame);

    cadre_actuel = im2double(frame);
    A = Soustraire_Background(cadre_actuel, Y, X, cadre_anterieur);
```

```
    subplot(2,2,2), imshow(A);
%drawnow;
    B = imdilate(A,ones(3));
    B = imfill(B,'holes');
    B = imerode(B,ones(3));
    B = medfilt2(A,[3 3]);
    subplot(2,2,4), imshow(B);


       % on calcule les limites de tous les blobs dans l'image
    boundaries = bwboundaries(A);
    bndrSize = size(boundaries);
    for k = 1:bndrSize(1)
      bnd = boundaries{k};
      RB2 = max(bnd);
      LT2 = min(bnd);
      if ( (max(RB2-LT2)>20) && notInsideOther(boundaries,k) )

        %extraire les caracteristiques
        [tr_object_ratio, tr_tete_ratio, tr_epaules_ratio , ...
         tr_tete_corps_ratio , tr_jambes_corps_ratio ,
tr_tronc_corps_ratio ] = ...
            Extraire_Caracteristiques(A, bnd);

        %Est-ce un humain? Appeler adaboost pour en savoir.
         te_labels(5,1) = 2;
         te_set(5,1) = 0;
         te_labels(:,1)=2;
        te_labels(1,1) = 1;

        te_set(1,1) = tr_object_ratio;
        te_set(1,2) = tr_tete_ratio;
        te_set(1,3) = tr_epaules_ratio;
        te_set(1,4) = tr_tete_corps_ratio;
        te_set(1,5) = tr_jambes_corps_ratio;
        te_set(1,6) = tr_tronc_corps_ratio;


        [L_te,hits_te] =
ADABOOST_te(adaboost_model,@threshold_te,te_set,te_labels);
        %Dessiner les carres autour des objets
        object_width = abs(RB2(2)-LT2(2));
        object_height = abs(RB2(1)-LT2(1));
%        hits_te = 1;
        hits_te = hits_te;
       hold on;

        if ((object_width~=0) && (object_height~=0))
            if (hits_te==5)
               subplot(2,2,3),
rectangle('Position',[LT2(2),LT2(1),object_width,object_height],'EdgeCo
lor','g','LineWidth',1);
```

```
                subplot(2,2,4),
rectangle('Position',[LT2(2),LT2(1),object_width,object_height],'EdgeCo
lor','g','LineWidth',1);
            else
                subplot(2,2,3),
rectangle('Position',[LT2(2),LT2(1),object_width,object_height],'EdgeCo
lor','r','LineWidth',1);
                subplot(2,2,4),
rectangle('Position',[LT2(2),LT2(1),object_width,object_height],'EdgeCo
lor','r','LineWidth',1);
            end
        end
        drawnow;
      end
    end
    aviframe = getframe(fig);
    aviobj = addframe(aviobj, aviframe);
    fp = fp+1;
end
aviobj = close(aviobj);

function m_inside = notInsideOther(m_boundaries, m_currelem)

    m_inside=true;
    bndrSize = size(m_boundaries);
    curbnd = m_boundaries{m_currelem};
    curRB2 = max(curbnd);
    curLT2 = min(curbnd);
    f = 0;
    % maintenant, on parcours tous les blobs qu'on a dans l'image
    for k = 1:bndrSize(1)
        if (m_currelem~=k)
          bnd = m_boundaries{k};
          RB2 = max(bnd);
          LT2 = min(bnd);
          if (  (   (curLT2(2)>LT2(2) && curLT2(2)<RB2(2)) && ...
                    (curLT2(1)>LT2(1) && curLT2(1)<RB2(1)) ) || ...
                (   (curRB2(2)>LT2(2) && curRB2(2)<RB2(2)) && ...
                (curRB2(1)>LT2(1) && curRB2(1)<RB2(1))  )   )
                m_inside=false;
                return
          end
        end
    end
```

8.3 Programme d'inisialisation de l'ensemble de training

Cette programme lit la feuille Excel avec les données d'entérinement et génère le modèle non linéaire

```
function adaboost_model = Init_Training_Set (source_donnees)
%
%
% Creating the training and testing sets
%


exl = actxserver('excel.application');

%% Load data from an excel file
% Get Workbook interface and open file
exlWkbk = exl.Workbooks;
exlFile = exlWkbk.Open([ source_donnees]);

% Get interface for Sheet1 and read data into range object
exlSheet1 = exlFile.Sheets.Item('Sheet1');
rngObj = exlSheet1.Range('A1:G140');

% Read data from excel range object into MATLAB cell array
exlData = rngObj.Value;
tr_tete_ratio = [];
tr_labels = [];
lBoxList = [];
mm = size(exlData,2)
%% Extract column data
for ii = 1:size(exlData,2)-1

   matData(:,ii) =
reshape([exlData{2:end,ii}],size(exlData(2:end,ii)));

end

tr_labels = reshape([exlData{2:end,7}],size(exlData(2:end,7)));

tr_object_ratio = matData(:,1);
tr_tete_ratio = matData(:,2);
tr_epaules_ratio = matData(:,3);
tr_tete_corps_ratio = matData(:,4);
tr_jambes_corps_ratio = matData(:,5);
tr_tronc_corps_ratio = matData(:,6);


tr_n = size(exlData,1);
%te_n = 200;
weak_learner_n = 20;

% Training and testing error rates
tr_error = zeros(1,weak_learner_n);
te_error = zeros(1,weak_learner_n);
```

```
for i=1:weak_learner_n
    adaboost_model =
ADABOOST_tr(@threshold_tr,@threshold_te,matData,tr_labels,i);
end

return;
```

8.4 Programme d'extraction des caracteristiques

```
function     [tr_object_ratio, tr_tete_ratio, tr_epaules_ratio , ...
     tr_tete_corps_ratio , tr_jambes_corps_ratio , tr_tronc_corps_ratio
] = Extraire_Caracteristiques(image_traite, bnd)


     % on calcule les limites de tous les blobs dans l'image
    %boundaries = bwboundaries(image_traite);
    %bndrSize = size(boundaries);

    tr_object_ratio = 0;
    tr_tete_ratio = 0;
    tr_epaules_ratio = 0;
    tr_tete_corps_ratio = 0;
    tr_jambes_corps_ratio = 0;
    tr_tronc_corps_ratio = 0;


    f = 0;
    % maintenant, on parcours tous les blobs qu'on a dans l'image
%    for k = 1:bndrSize(1)
%      bnd = boundaries{k};
      RB2 = max(bnd);
      LT2 = min(bnd);
      if ( (max(RB2-LT2)>30)  )
          % on calcule l'hauteur, la longueur et le ratio
hauteur/longueur
          object_width = abs(RB2(2)-LT2(2));
          object_height = abs(RB2(1)-LT2(1));
          tr_object_ratio = object_height/object_width;

          % alors je divise en trois: Tete, tronc, et jambes.
          lim_tete = object_height/4;
          lim_tronc = lim_tete+(object_height/3);
          lim_jambes = lim_tronc+(object_height-lim_tronc);
          c_tete = [LT2(2), LT2(2), LT2(2)+object_width,
LT2(2)+object_width];
          r_tete = [LT2(1), LT2(1)+lim_tete,LT2(1)+lim_tete,LT2(1)];

          c_tronc = [LT2(2), LT2(2), LT2(2)+object_width,
LT2(2)+object_width];
          r_tronc = [LT2(1)+lim_tete,
LT2(1)+lim_tronc,LT2(1)+lim_tronc,LT2(1)+lim_tete];
```

```
          c_jambes = [LT2(2), LT2(2), LT2(2)+object_width,
LT2(2)+object_width];
          r_jambes = [LT2(1)+lim_tronc,
LT2(1)+lim_jambes,LT2(1)+lim_jambes,LT2(1)+lim_tronc];

          % je calcule les images pour chaque section
          FR = im2double(image_traite);
          ROI_TETE = im2double ( roipoly(image_traite,c_tete,r_tete) );
          ROI_TRONC = im2double ( roipoly(image_traite,c_tronc,r_tronc)
);
          ROI_JAMBES = im2double (
roipoly(image_traite,c_jambes,r_jambes) );

          tete = FR.*ROI_TETE;
          tronc = FR.*ROI_TRONC;
          jambes = FR.*ROI_JAMBES;


          f = f+1;
            %relation du tete
          minvalue = 999;
            maxvalue = 0;
            linemin=0;
            linemax=0;
            for line_tete=LT2(1)+15:LT2(1)+lim_tete
               cursize = 0;
               for col_tete = LT2(2):LT2(2)+object_width
                   value = tete(line_tete,col_tete);
                   if (value>.9)
                       cursize=  cursize+1;
                   end
               end
               if (cursize>10)
               if (cursize>maxvalue)
                   maxvalue = cursize;
                   linemax = line_tete;
               end
               if (cursize<minvalue)
                   minvalue = cursize;
                   linemin = line_tete;
               end
               end
            end
            tr_tete_ratio = maxvalue / minvalue;
            epaules = maxvalue;
            %epaules
            minvalue = 999;
            maxvalue = 0;
            linemin=0;
            linemax=0;
            numlines = 0;
            for line_tronc=LT2(1)+round(lim_tete)+1:LT2(1)+lim_tronc
               numlines = numlines + 1;
               cursize = 0;
               for col_tronc = LT2(2):LT2(2)+object_width
```

```
                    value = tronc(line_tronc,col_tronc);
                    if (value>.9)
                        cursize=  cursize+1;
                    end
                end
                maxvalue = maxvalue+cursize;
            end
            moyenne_tronc = maxvalue/numlines;

            tr_epaules_ratio = moyenne_tronc/epaules;
            %autres ratios
            tete_surface = bwarea(tete);
            tronc_surface = bwarea(tronc);
            jambes_surface = bwarea(jambes);
            object_surface = tete_surface+tronc_surface+jambes_surface;

            tr_tete_corps_ratio = tete_surface / object_surface;
            tr_jambes_corps_ratio = jambes_surface / object_surface;
            tr_tronc_corps_ratio = tronc_surface / object_surface;

        end
%     end
```

8.5 Programme d'analyse des exemples et génération du feuille Excel

```
function GenererEnsembleExample()

% Ouverture du fichier video et extraction des caracteristiques
% (taille de l'image, nombre de cadres)


m_pathsource = 'C:\Pablo\University\Hiver 2010\Projet
Synthese\Images\Examples\Mauvaises\traitees\';
m_pathdest = 'C:\Pablo\University\Hiver 2010\Projet
Synthese\Images\Examples\Mauvaises\';
m_fichiercaracteristiques = strcat(m_pathdest,'mauvaises.xls');
m_pathsource = 'C:\Pablo\University\Hiver 2010\Projet
Synthese\Images\Examples\Bonnes\traitees\';
m_pathdest = 'C:\Pablo\University\Hiver 2010\Projet
Synthese\Images\Examples\Bonnes\';
m_fichiercaracteristiques = strcat(m_pathdest,'bonnes.xls');
linefile = 0;
mymatrix = [];
files = dir(m_pathsource )
numfiles = size(files,1)
fp=3;
while (fp<numfiles+1)
    if (files(fp).isdir)
        fp = fp+1;
        continue
    end
    m_Image = files(fp).name;
```

```
    %m_Image = 'frame_0000.jpg';
    %frame_0004
    %frame_0024
    %if (strfind(m_Image,'jpg')==[])
    %    f = f+1;
    %    continue
    %end
    m_fichierImage = strcat(m_pathsource,m_Image);

    A = imread(m_fichierImage);

    subplot(2,3,1),    imshow(A);

    % on calcule les limites de tous les blobs dans l'image
    boundaries = bwboundaries(A);
    bndrSize = size(boundaries);

    f = 0;
    % maintenant, on parcours tous les blobs qu'on a dans l'image
    for k = 1:bndrSize(1)
      bnd = boundaries{k};
      RB2 = max(bnd);
      LT2 = min(bnd);
      if ( (max(RB2-LT2)>30)    && notInsideOther(boundaries,k) )
          % on calcule l'hauteur, la longueur et le ratio
hauteur/longueur
          object_width = abs(RB2(2)-LT2(2));
          object_height = abs(RB2(1)-LT2(1));
          object_ratio = object_height/object_width;

            %% je trace un rectangle dans la figure originale
            subplot(2,3,1),
%            if ((object_width>0) && (object_height>0))

rectangle('Position',[LT2(2),LT2(1),object_width,object_height],'EdgeCo
lor','g','LineWidth',1);
 %           end
            % alors je divise en trois: Tete, tronc, et jambes.
            lim_tete = object_height/4;
            lim_tronc = lim_tete+(object_height/3);
            lim_jambes = lim_tronc+(object_height-lim_tronc);
            c_tete = [LT2(2), LT2(2), LT2(2)+object_width,
LT2(2)+object_width];
            r_tete = [LT2(1), LT2(1)+lim_tete,LT2(1)+lim_tete,LT2(1)];

            c_tronc = [LT2(2), LT2(2), LT2(2)+object_width,
LT2(2)+object_width];
            r_tronc = [LT2(1)+lim_tete,
LT2(1)+lim_tronc,LT2(1)+lim_tronc,LT2(1)+lim_tete];

            c_jambes = [LT2(2), LT2(2), LT2(2)+object_width,
LT2(2)+object_width];
            r_jambes = [LT2(1)+lim_tronc,
LT2(1)+lim_jambes,LT2(1)+lim_jambes,LT2(1)+lim_tronc];
```

```
% je calcule les images pour chaque section
FR = im2double(A);
ROI_TETE = im2double ( roipoly(A,c_tete,r_tete) );
ROI_TRONC = im2double ( roipoly(A,c_tronc,r_tronc) );
ROI_JAMBES = im2double ( roipoly(A,c_jambes,r_jambes) );

tete = FR.*ROI_TETE;
tronc = FR.*ROI_TRONC;
jambes = FR.*ROI_JAMBES;


f = f+1;
m_col = (f*3);
%%Et je montre mes images
subplot(2,3,4), imshow(tete);
subplot(2,3,5), imshow(tronc);
subplot(2,3,6), imshow(jambes);
% relation de la tete

%relation du tronc


%relation des jambes?
%          subplot(3,3,7), imshow(imerode(tete, ones(3)) );
%          subplot(3,3,7), imshow(tete);
minvalue = 999;
maxvalue = 0;
linemin=0;
linemax=0;
for line_tete=LT2(1)+15:LT2(1)+lim_tete
   cursize = 0;
   for col_tete = LT2(2):LT2(2)+object_width
       value = tete(line_tete,col_tete);
       if (value>.9)
           cursize=  cursize+1;
       end
   end
   if (cursize>10)
   if (cursize>maxvalue)
       maxvalue = cursize;
       linemax = line_tete;
   end
   if (cursize<minvalue)
       minvalue = cursize;
       linemin = line_tete;
   end
   end
end
tete_ratio = maxvalue / minvalue;
epaules = maxvalue;

minvalue = 999;
maxvalue = 0;
linemin=0;
```

```
            linemax=0;
            numlines = 0
            for line_tronc=LT2(1)+round(lim_tete)+1:LT2(1)+lim_tronc
               numlines = numlines + 1;
               cursize = 0;
               for col_tronc = LT2(2):LT2(2)+object_width
                   value = tronc(line_tronc,col_tronc);
                   if (value>.9)
                       cursize=  cursize+1;
                   end
               end
               maxvalue = maxvalue+cursize
            end
            moyenne_tronc = maxvalue/numlines;

            ratio_tronc_epaules = moyenne_tronc/epaules




            tete_surface = bwarea(tete);
            tronc_surface = bwarea(tronc);
            jambes_surface = bwarea(jambes);
            object_surface = tete_surface+tronc_surface+jambes_surface;
            ratio_tete_obj = tete_surface / object_surface;
            ratio_jambes_obj = jambes_surface / object_surface;
            ratio_tronc_obj = tronc_surface / object_surface;

            linefile = linefile+1;
            mymatrix(linefile,:) = [object_ratio, tete_ratio,
ratio_tronc_epaules, ratio_tete_obj, ratio_jambes_obj,
ratio_tronc_obj];

          drawnow
      end
    end
%    pause;
    fp = fp+1;

end
xlswrite (m_fichiercaracteristiques,mymatrix);


function m_inside = notInsideOther(m_boundaries, m_currelem)

    m_inside=true;
    bndrSize = size(m_boundaries);
    curbnd = m_boundaries{m_currelem};
    curRB2 = max(curbnd);
    curLT2 = min(curbnd);
    f = 0;
    % maintenant, on parcours tous les blobs qu'on a dans l'image
    for k = 1:bndrSize(1)
```

```
        if (m_currelem~=k)
          bnd = m_boundaries{k};
          RB2 = max(bnd);
          LT2 = min(bnd);
          if (  (   (curLT2(2)>LT2(2) && curLT2(2)<RB2(2)) && ...
                    (curLT2(1)>LT2(1) && curLT2(1)<RB2(1)) ) || ...
                (   (curRB2(2)>LT2(2) && curRB2(2)<RB2(2)) && ...
                (curRB2(1)>LT2(1) && curRB2(1)<RB2(1))  )   )
                m_inside=false;
                return
          end
        end
    end
```

8.6 Programme de soustraction de fond

```
function A = Soustraire_Background(cadre_actuel, image_background,
image_normalisee, cadre_anterieur)

thresh_dbl = .2;
thresh_byte = .06;   %250;
h = ones(5,5) / 25;

if (0)
V = Normaliser_Image(cadre_actuel);
U = imabsdiff( rgb2hsv(cadre_actuel), rgb2hsv(image_background));
U(:,:,2)=0;
U (U<0.13) = 0;
U (U>=0.13) = 1;

U (U<0.13) = 0;
U (U>=0.13) = 1;
PO = U(:,:,3);
PO = medfilt2(PO,[3 3]);
%PO = imdilate(PO,ones(5));
PO = imfill(PO,'holes');
%subplot (2,2,4), imshow(PO);
end
img_diff = imabsdiff (cadre_actuel,image_background);

img_gray1 = rgb2gray(img_diff);


A = img_gray1;


A(A<thresh_byte)=0;
A(A>=thresh_byte)=255;
A = medfilt2(A,[3 3]);
A=  imfill(A,'holes');
```

```
return;


function Y = UpdateBackground(Current_Background,New_Image)
 %   U (U<0.31) = 0;
Current_Background(Current_Background>New_Image) =
Current_Background(Current_Background>New_Image) - 0.0020;
Current_Background(Current_Background<New_Image) =
Current_Background(Current_Background<New_Image) + 0.0010;

Y = Current_Background;

function Y = CalculerMoyenne(RGB_Background, K)

Y = RGB_Background(:,:,:,1);

for cadres = 2:K
    Y = imadd(Y,RGB_Background(:,:,:,cadres));
end
Y = imdivide(Y,K);
```