Table des matières

Introduction

Dans le cadre de ce projet, l'essentiel de ce qu'il y a à réaliser est de suivre un objet dans une séquence d'images (autrement dit, une vidéo); pour ce faire, il faudra implanter l'algorithme Mean-Shift, qui est un moyen efficace de faire précisément cela. Pour y parvenir, il faut s'abord acquérir des connaissances de base dans le traitement numérique d'images.

En résumé, l'algorithme Mean-Shift tente de trouver, à partir d'une image initiale, la zone d'une image cible qui lui correspond en comparant les fréquences de chaque intensité de couleur présente (ou les fréquences des niveaux de gris dans le cas d'une image en noir et blanc). Il s'agit d'une méthode statistique qui s'avère, en théorie, d'une exactitude plus que suffisante et d'une grande rapidité.

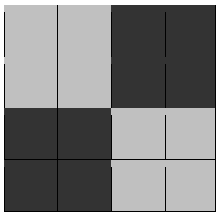
Ce document se veut un résumé de ce qui a été fait durant la session dans le cadre de ce projet; des explications sommaires, quelques extraits de code et figures détailleront brièvement le travail réalisé ainsi que les étapes franchies.

Tout ce qui va être présenté comme algorithme dans ce document a été implémenté avec Matlab 7.1.

Chapitre I : Notions préalables

Images, matrices et histogrammes

Une image peut être représentée de façon matricielle. Par exemple, on pourrait représenter une simple image de niveaux de gris 4x4 ayant grossièrement cette apparence

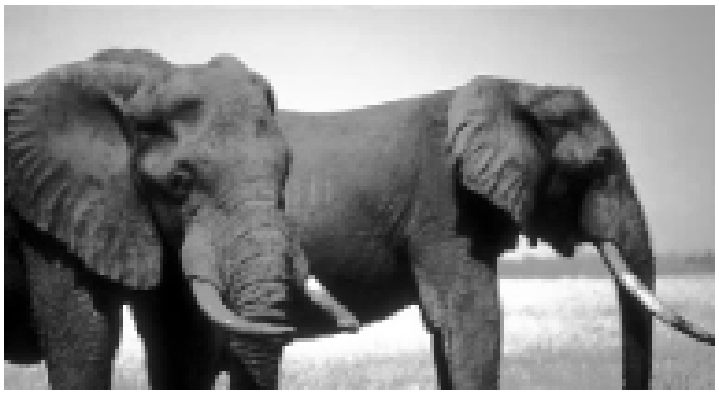


par la matrice bidimensionnelle suivante :

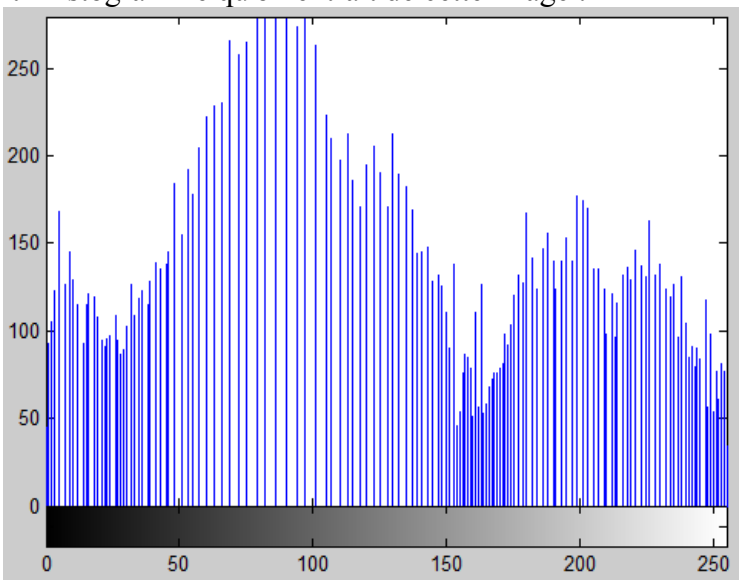
$$I = \begin{matrix} 200 & 200 & 80 & 80 \\ 200 & 200 & 80 & 80 \\ 80 & 80 & 200 & 200 \\ 80 & 80 & 200 & 200 \end{matrix}$$

Dans ce cas-ci, l'image est représentée selon une échelle de gris de huit bits (un byte), c'est-à-dire que chaque valeur de gris variera entre 0 et 255. Une valeur de zéro dénote une intensité de blanc nulle (donc un noir total) et une valeur de 255 représente au contraire un blanc complet. De façon analogue, lorsqu'on a une image en couleurs, chaque pixel est représenté par trois bytes : un pour le rouge, un pour le vert et un pour le bleu. Pour les besoins de ce documents et pour simplifier les explications, seules des images noir et blanc seront utilisées.

Il est également intéressant de consulter l'histogramme d'une image, surtout lorsque l'on souhaite voir dans quelles intensités se concentrent surtout ses pixels. L'histogramme aura comme abcisses l'ensemble des valeurs prises par les pixels de l'image et comme ordonnées la fréquence de ces valeurs (autrement dit, le nombre de fois que chaque valeur appraraît dans l'image). Voici un exemple d'image :



Voici maintenant l'histogramme qu'on extrait de cette image :



Négatif d'une image

Obtenir le négatif d'une image numérique est une manipulation très simple : la valeur de chaque pixel de l'image transformée sera égale à 255 (dans le cas d'une image huit bits) moins la valeur du même pixel dans l'image originale. L'algorithme qui en résulte est le suivant :

```
I = imread('image_originale.pgm');

for b = 1:192
    for a = 1:104
        I(a, b) = 255 - I(a, b);
    end
end

imwrite(I, 'image_neg.pgm', 'pgm');
```

Cet exercice, bien qu'il ne servira pas explicitement dans l'implémentation de l'algorithme Mean-Shift, peut être d'une grande aide dans la compréhension de la structure des données d'une image numérique ainsi que dans son traitement.

Amélioration de qualité d'une image

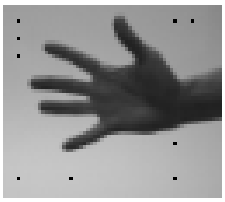
Avant de tenter de repérer un objet dans une image ou encore de le suivre dans une vidéo, il est très important de s'assurer d'avoir une image de qualité et dont les caractéristiques ne nous empêchent pas d'effectuer une analyse approfondie. À ces fins, plusieurs techniques ont été élaborées pour corriger, par exemple, les défauts que peuvent contenir une image, ou encore ajuster son contraste pour que les détails paraissent plus nets.

Élimination du bruit

Les images bruitées, c'est-à-dire qui comportent des pixels trop différents de leurs voisins immédiats, peuvent être corrigées grâce à une méthode simple qui calcule la moyenne d'un groupe de pixels pour redéfinir la valeur du pixel du milieu du groupe. En diminuant l'écart entre chaque pixel et tous ses voisins, on s'assure qu'aucun ne ressorte trop nettement de l'image. Voici, en exemple, un algorithme qui améliore la qualité d'une image en réduisant son bruit :

```
I = imread('..\images\image_bruit.pgm');
I_s = size(I);

I2 = I;
for y = 2:I_s(1)-1
    for x = 2:I_s(2)-1

        moy = uint16(0);
        for v = -1:1:1
            for u = -1:1:1
                moy = moy + uint16(I(y+v, x+u));
            end
        end
        moy = round(moy/9);
        I2(y,x) = uint8(moy);

    end
end

imwrite(I2, '..\images\image_corr.pgm', 'pgm');
```

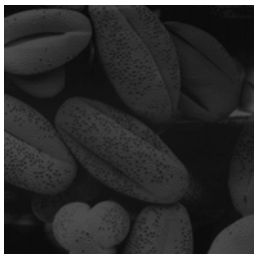
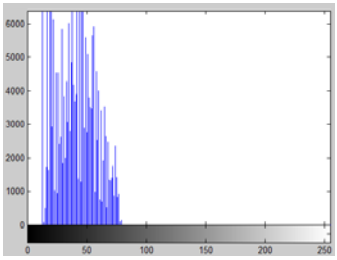
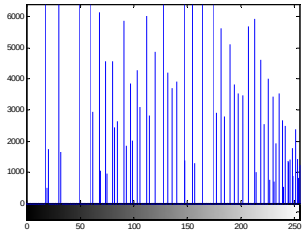
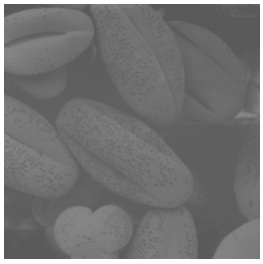
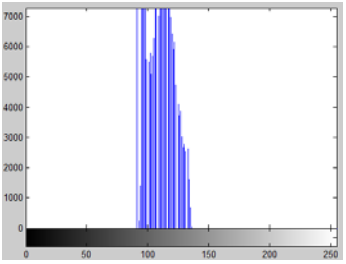
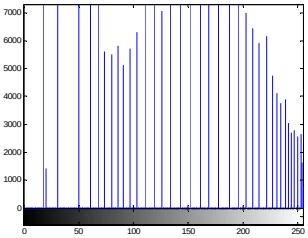
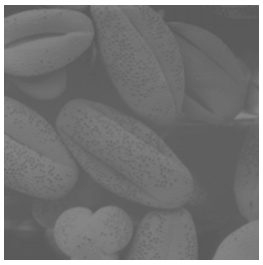
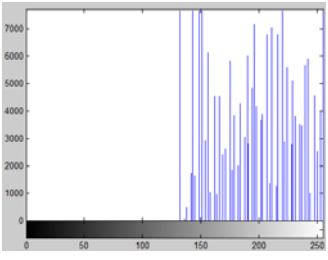
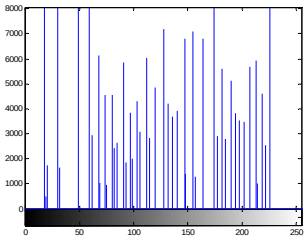
Voici une comparaison entre l'image avant le traitement pour éliminer le bruit et l'image qui résulte du traitement :



Il va de soi que ce traitement est somme toute rudimentaire, mais en l'appliquant à une image de plus grandes dimensions ou en utilisant un masque plus grand (le masque utilisé ici était de 3x3) le résultat obtenu pourrait paraître moins flou. Néanmoins, les points noirs présents sur l'image originale sont estompés, presque effacés sur l'image traitée et cela va aider à aplanir l'histogramme de cette dernière.

Égalisation d'histogramme

L'égalisation d'histogramme est un autre traitement qu'il est intéressant d'appliquer à une image [1]. Il sera opportun d'égaliser l'histogramme d'une image lorsque celle-ci sera trop foncée, trop pâle ou dont les tons de gris (dans le cas d'une image noir et blanc) sont trop semblables, pas assez distincts les uns des autres.

Image	Histogramme original	Histogramme égalisé
		
		
		

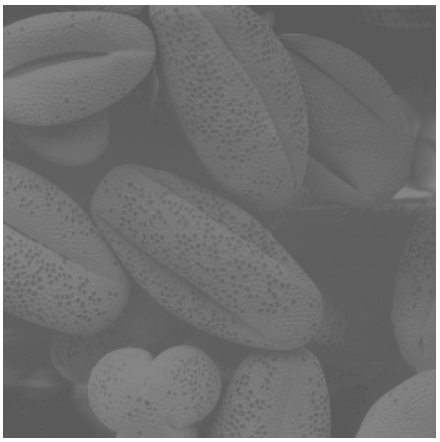
Cette méthode vise à répartir les intensités dans l'histogramme proportionnellement à leur fréquence, utilisant la fonction de répartition; bien que les histogramme égalisés des trois images du tableau précédent semblent passablement différents, les trois images modifiées selon cette méthode seront très semblables. Voici un algorithme qui permet d'effectuer cette opération :

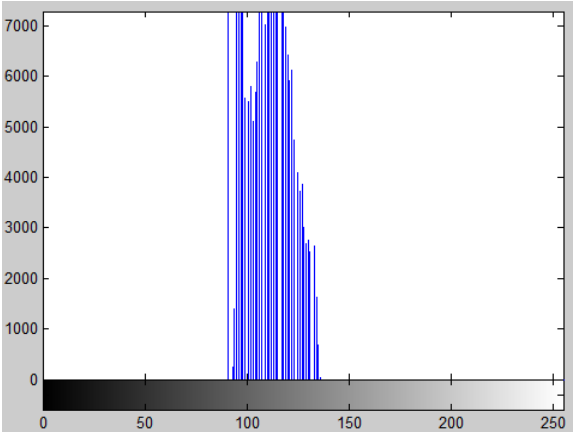
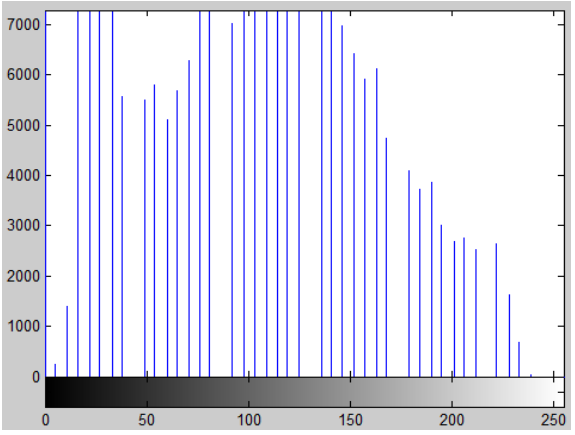
```
I = imread('..\images\image.pgm');
R = imhist(I);
tempSomme = 0;
S = size(I);

for i = 1:256
    tempSomme = tempSomme + R(i);
    f_repartition(i) = tempSomme / (S(1) * S(2));
end

for x = 1:S(2)
    for y = 1:S(1)
        I2(y,x) = round(255 * f_repartition(I(y,x)+1));
    end
end
I2 = uint8(I2);

imwrite(I2, '..\images\image_eq.pgm', 'pgm');
```

Comme l'indique ce code, la nouvelle valeur de chaque pixel de la nouvelle image sera proportionnelle à la probabilité cumulative qu'avait la valeur du même pixel dans l'image originale. Par exemple, dans l'image d'éléphants présentée plus haut, un pixel quelconque a 59,38 % de chances d'avoir une valeur inférieure ou égale à 150; en multipliant 0,5938 par 255 (la valeur maximale que peut prendre un pixel) on obtient 151,419, qu'on arrondit à 151. Ainsi, un pixel ayant une valeur de 150 dans l'image originale aura une valeur de 151 dans l'image modifiée.

Changement linéaire de dynamique

Il existe d'autres méthodes qui visent à égaliser l'histogramme d'une image, notamment le changement de dynamique linéaire. Dans cette méthode, il s'agit simplement d'étirer un histogramme en tirant son minimum à 0 et en repoussant son maximum à 255.

Image originale	Image modifiée
	

Histogramme original	Histogramme modifié
	

On peut remarquer que les barres de fréquences dans l'histogramme modifié sont pour la plupart équidistantes, respectant leurs distances relatives dans l'histogramme original. L'algorithme qui permet d'effectuer cette opération est le suivant :

```
I = imread('..\images\image_mid.pgm');
I_s = size(I);
H = imhist(I);

for i = 1:256
   if H(i) > 0
      min = double(i);
      break;
   end
end

for i = 256:-1:1
   if H(i) > 0
      max = double(i);
      break;
   end
end

for y = 1:I_s(1)
   for x = 1:I_s(2)
      J(y,x) = round(255 * double((I(y,x) - min)) / double((max - min)));
   end
end
J = uint8(J);

imwrite(J, '..\images\image_dyn.pgm', 'pgm');
```

Essentiellement très simple, cet algorithme détermine tout d'abord le minimum et

le maximum de l'histogramme initial, puis calcule la valeur de chaque pixel de l'image résultante. L'image ainsi modifiée paraîtra très semblable à une image modifiée par l'algorithme d'égalisation décrit auparavant, mais les histogrammes des deux images montreront des différences évidentes lorsque consultés de plus près.

Corrélation d'image

Lorsque les images à traiter ont une apparence et une forme appropriées, il devient possible de rechercher l'image d'un objet dans une plus grande image qui contient cet objet. La corrélation par filtre (ou par masque) offre une alternative à l'algorithme Mean-Shift facile à implémenter et à comprendre; cela dit, comme il sera manifeste de par l'algorithme, il s'agit d'une méthode très coûteuse en temps de calcul.

L'algorithme qui suit est amplement commenté pour aider à sa compréhension :

```
% Lire les images
I = imread('..\images\petite_image.pgm', 'pgm');
J = imread('..\images\grande_image.pgm', 'pgm');

% Initialiser les dimensions des images
I_s = size(I);
J_s = size(J);

% Initialiser cos theta maximal au minimum possible
max = intmin('uint32');

% Calculer la magnitude du vecteur I (petite image)
I_mag = double(0);
for b = 1:I_s(1)
    for a = 1:I_s(2)
        I_mag = I_mag + double(I(b,a)) * double(I(b,a));
    end
end
I_mag = sqrt(I_mag);

% Parcourir la matrice J (grande image) en calculant le produit scalaire
% des matrices et la magnitude de chaque cadre
for y = 1:( J_s(1) - I_s(1) )
    for x = 1:( J_s(2) - I_s(2) )

        % Calcul du produit scalaire des deux matrices
        tempval = uint32(0);
        J_mag = double(0);
        for v = 1:I_s(1)
            for u = 1:I_s(2)
                tempval = tempval + uint32( J(y+v-1,x+u-1) ) * uint32 ( I(v,u) );
                J_mag = J_mag + double(J(y+v-1,x+u-1)) * double(J(y+v-1,x+u-1));
            end
        end

        % Calcul de la magnitude et du cosinus de l'angle obtenu
        J_mag = sqrt(J_mag);
        map(y, x) = double( double(tempval) / double(I_mag * J_mag) );

        % S'il s'agit d'un nouveau maximum, l'enregistrer ainsi que
        % ses coordonnées
        if map(y, x) > max
            max = map(y, x);
            coords = [y, x];
        end

    end
end

% Affichage des résultats
disp('Image (y, x) trouvée à: ');
disp(coords);
disp('Magnitude maximale: ');
disp(max);
```

Dans cet algorithme, l'essentiel des opérations consiste à calculer les produits scalaires [2] entre différents vecteurs. Mathématiquement, une matrice bidimensionnelle (dans ce cas-ci, la matrice d'une image) peut aussi être représentée sous la forme d'un vecteur unidimensionnel si on aligne toutes les rangées pour n'en former qu'une seule. Sachant qu'on peut passer d'une image à un vecteur aisément, comparer deux images revient tout simplement à comparer deux vecteurs. Or, la formule mathématique suivante nous permet de faire précisément cela :

$$\cos(\theta) := \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}| \cdot |\mathbf{B}|}$$

Dans cette formule, A et B sont deux vecteurs non nuls et θ est l'angle qu'ils décrivent. Si un angle de 0 signifie que les deux vecteurs sont égaux (donc que deux images qu'on compare ainsi sont identiques), c'est donc à dire qu'on cherche un cosinus d'angle maximal résultant du produit scalaire de deux vecteurs divisé par le produit de leurs magnitudes (ou longueurs) pour trouver la paire de vecteurs qui se ressemblent le plus. Posant le vecteur A comme étant le vecteur obtenu de l'image recherchée, il ne reste qu'à calculer l'angle obtenu en le comparant avec tous les vecteurs B que l'on peut obtenir à partir de la plus grande image, celle dans laquelle on cherche, puis déterminer avec quel B on a obtenu un résultat maximal, c'est-à-dire un angle le plus près possible de 0.

Comme mentionné plus haut, cet algorithme est très exigeant en temps de calcul. Prenons par exemple la recherche d'une image de dimensions 50x50 dans une image de dimensions 250x250, en supposant que ces deux images sont en noir et blanc pour simplifier les calculs; d'autre part, ne seront calculées que les additions (A), les multiplications (M) et les divisions (D), en supposant que la lecture initiale des deux images, les assignations de variables et les quelques autres opérations diverses sont instantanées ou ne dépendent pas des dimensions :

1. La magnitude de la petite image est d'abord calculée; celle-ci ne change pas et n'est donc calculée qu'une seule fois.

   Coût : 50 * 50 = 2500 A et 2500 M

2. Il faut maintenant parcourir la grande image à la recherche de la petite. À chaque « cadre » ainsi délimité, il faudra calculer le produit scalaire avec la petite image.

   Coût : (250-50) * (250-50) = 40 000 cadres, 2500 A et 2500 M par cadre
   100 000 000 A et 100 000 000 M

3. À chaque cadre correspond également une magnitude, nécessaire aux calculs.

   Coût : 40 000 cadres, 2500 A et 2500 M par cadre
   100 000 000 A et 100 000 000 M

4. Finalement, le cosinus de l'angle décrit par les deux vecteurs est calculé.

   Coût : 40 000 cadres, 1 M et 1 D par cadre
   40 000 M et 40 000 D

Coût total approximatif : 200 002 500 A, 200 042 500 et 40 000 D = 400 085 000 opérations.

Avec plus de 400 millions d'opérations nécessaires pour comparer de si petites images noir et blanc, il appert évident qu'un tel algorithme ne peut pas, selon nos moyes actuels, fonctionner en temps réel; de plus, son application se ferait sur un contenu vidéo, où jusqu'à plusieurs dizaines d'images à la seconde devraient être traitées.

De plus, cet algorithme ne fonctionne que s'il y a un alignement parfait de l'objet référence (masque) avec la partie candidate. La corrélation est sensible au changement d'orientation, au changement d'échelle ainsi qu'au bruit.

En somme, bien que théoriquement cet algorithme trouve toujours la meilleure correspondance entre deux images, sa robustesse lacunaire et sa grande lenteur font en sorte qu'il ne pourra pas être utilisé.

Chapitre II : L'algorithme Mean Shift

Fonctionnement

Pour trouver efficacement une image dans une autre, il faut se tourner vers des algorithmes beaucoup plus performants; Mean-Shift est l'un d'eux. Usant d'astuces mathématiques et tenant compte de la notion qu'un objet en mouvement dans une vidéo ne sera jamais très éloigné d'une image (ou d'un cadre) à l'autre. Cette dernière section présentera quelques détails de l'implémentation de l'algorithme Mean-Shift.

Tout d'abord, Mean-Shift opère de façon similaire à l'algorithme précédent, c'est-à-dire qu'il doit également comparer deux images à partir desquelles on a calculé des vecteurs de caractéristiques [3] auparavant. La fonction à la base de cette comparaison s'appelle le coefficient de Bhattacharyya; il est défini comme suit :

$$B(p,q) := \sum_{u=1}^{m} \sqrt{p(u)\cdot q(u)}$$

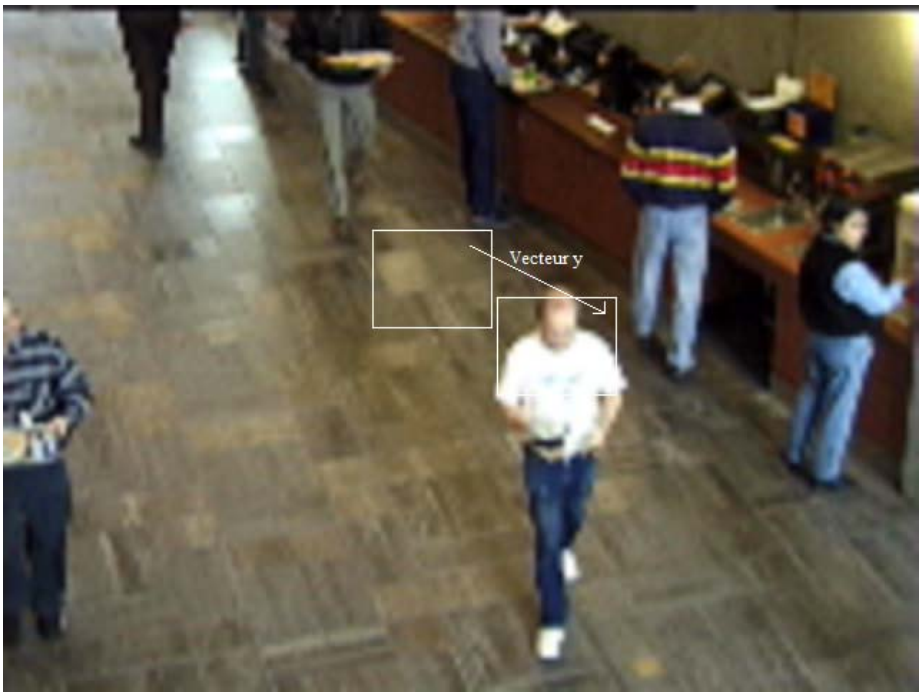
Le coefficient de Bhattacharyya est typiquement appliqué à deux distributions probabilistes pour mesure leur similarité; dans ce cas-ci, les deux distributions en questions sont les fréquences relatives associées au candidat (p) ainsi qu'à la référence (q). De plus, un coefficient égal à 1 signifiera que les deux distributions sont identiques alors qu'un coefficient égal à 0 indiquera qu'elles n'ont absolument aucune ressemblance. Le but ici est donc de trouver le candidat qui maximise le coefficient de Bhattacharyya lorsque comparé avec la référence.

Bien que le coefficient de Bhattacharyya soit un bon outil pour comparer les deux distributions (et par conséquent les deux images), un important problème demeure : pour trouver le meilleur candidat avec cette méthode, il faudra nécessairement comparer tous les autres candidats. Du point de vue de la complexité algorithmique de cette procédure, nous nous trouvons encore dans le voisinage de celle de la corrélation d'image (voir section précédente).

La forme particulière (et adaptée à notre problème) du coefficient de Bhattacharyya est écrite comme suit :

$$B(p(y),q) := \sum_{u=1}^{m} \sqrt{p_u(y)\cdot q_u}$$

où la variable y nous permet d'ajouter un vecteur (ou déplacement linéaire dans le plan de l'image) au candidat. Le fait de paramétrer l'équation nous permettra de remplacer une recherche exhaustive (dont la complexité algorithmique est équivalente à la corrélation d'image) parmi tous les candidats par une solution analytique qui peut être résolue algébriquement; cette dernière solution consiste à procéder à une estimation du coefficient de Bhattacharyya en passant par un développement de Taylor. On pourrait illustrer ce vecteur ainsi :



Après maintes manipulations algébriques [4], l'estimation d'un vecteur bidimensionnel menant à un meilleur candidat est ainsi effectuée, supposant que le candidat est composé des pixels x_1, x_2, ..., x_n et que les coordonnées de ces points sont prises par rapport au repère dont l'origine est le centre du candidat :

$$y_1 := \frac{\sum_{i=1}^{n} x_i w_i}{\sum_{i=1}^{n} w_i}$$

$$w_i := \sum_{u=1}^{m} \sqrt{\frac{q_u}{p_u(y_0)}} \cdot \delta\left(b(x_i) - u\right)$$

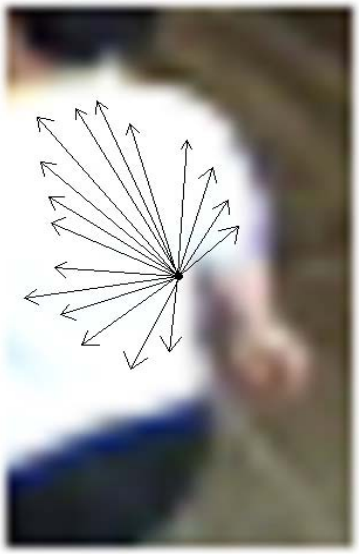
où y_1 est le vecteur calculé, x_i est un point du candidat, y_0 est le vecteur estimé initial (normalement nul) et δ est la fonction delta de Kronecker qui est ainsi définie :

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0. \end{cases}$$

Un pseudo-code de l'algorithme résultant irait comme suit :

```
Lire image
       y = [0, 0]
       Lire candidat selon y
       y1 = meanshift(candidat)
       Compteur = 0
       Faire
              Compteur = Compteur + 1
              y = y + y1
              Lire candidat selon y
              y1 = meanshift(candidat)
       Jusqu'à :    Compteur >= 10
                    OU longueur(y1) < Seuil
                    OU B(reference, candidat) ≈ 1
       y = y + y1
       Lire candidat selon y
       Tracer un cadre autour du candidat
       Afficher/enregistrer l'image résultante
Image Suivante
```

Ainsi, au lieu de chercher à trouver le meilleur candidat parmi tous les candidats possibles, ce qui serait beaucoup trop coûteux en temps, on estime un vecteur qui nous mènera à un *meilleur* candidat, à proximité de celui à partir duquel on effectue nos calculs. Même si une seule itération de ce calcul ne nous donne pas le résultat que l'on recherche, il n'est nécessaire que d'un petit nombre d'itérations pour trouver le candidat ressemblant le plus à la référence.



Ce processus est ainsi répété jusqu'à ce que le coefficient de Bhattacharyya de la référence et du

candidat (obtenu en se déplaçant selon le vecteur calculé) satisfasse un seuil prédéterminé, jusqu'à ce que la longueur du vecteur de déplacement calculé soit jugée négligeable ou jusqu'à ce qu'on ait itéré un certain nombre de fois (typiquement, environ une dizaine de fois).

Commentaires sur l'implémentation

Dans une optique où on cherche à effectuer du suivi d'objets en temps réel, l'algorithme Mean Shift se révèle aussi rapide en pratique qu'il le paraît sur papier. Pour accomplir cette rapidité, plusieurs optimisations ont été nécessaires; en effet, implémenter tel quel l'algorithme se révèle plutôt lent sur un ordinateur conventionnel, puisque plusieurs calculs se répètent inutilement et alourdissent grandement l'exécution. Des optimisations supplémentaires pourraient être apportées, par exemple, en convertissant l'algorithme en code C++; cela dit, plusieurs fonctions très utiles et disponibles dans l'environnement utilisé (Matlab) devront être réécrites.

Résultats

Voici une séquence d'images résultant de cette implémentation de l'algorithme Mean Shift avec comme référence l'image suivante :

Perspectives

Plusieurs améliorations et ajouts peuvent être apportés à cette implémentation de l'algorithme Mean Shift; cette partie présentera la gestion d'arrière-plan, l'intégration de différents types de couleur, l'affichage de la trajectoire 3D et la détection de cibles.

Gestion d'arrière-plan

Lors du choix d'une référence, tous les pixels de cette dernière vont contribuer à la définir; à priori, ce principe semble donner de bons résultats, mais dans les faits, plusieurs éléments peuvent venir brouiller la qualité de la référence. Un de ces éléments, qui se retrouve presque toujours dans une référence, est l'arrière-plan. Si l'arrière-plan occupe une partie trop importante de la référence, il est possible que la recherche d'un candidat dans le voisinage de cet arrière-plan produise un faux positif.

Le but de cette caractéristique que l'on souhaite intégrer à l'algorithme sera donc d'inhiber la contribution des pixels de la référence ressemblant à l'arrière-plan de l'image; dans le cas d'une séquence d'images passablement statique, telle une vidéo de surveillance, établir ce que constitue l'arrière-plan (plancher, mur, comptoir, panneau d'affichage, etc.) est une opération qui ne doit à toutes fins pratiques être effectuée qu'une seule fois.

Intégration de différents types de couleur

Il va sans dire que plus une référence se distingue de ce qui l'entoure, par exemple par ses contrastes, plus son suivi sera facile. Un élément qui a volontairement été omis dans cette implémentation de l'algorithme Mean Shift, la couleur, aide beaucoup à caractériser toute image; essentiellement, un pixel en couleurs possède trois fois plus d'information qu'un pixel de niveaux de gris.

Si par couleur on entend habituellement les longueurs d'ondes de la lumière visible (c'est-à-dire le rouge, le jaune, le bleu, etc.), le fait est que les rayonnements électromagnétiques forment une bande beaucoup plus large qu'il n'en paraît. En effet, en plus des couleurs visibles par l'oeil humain, il est possible, avec des instruments spécialisés, de percevoir d'autres fréquences du spectre électromagnétique. En particulier, le rayonnement infrarouge des objets peut servir à plusieurs applications, des équipements de vision de nuit à la thermographie infrarouge, en passant par les détecteurs de faux billets.

Affichage de la trajectoire 3D

Cette fonctionnalité consisterait à effectuer le suivi d'objets dans une séquence d'images (ou dans une vidéo) puis à aligner chaque image de façon à pouvoir tracer le déplacement dans le temps des objets ainsi suivis. Ce processus, comme l'indique son nom, nous donnera en quelque sorte la trajectoire de l'objet. Ceci pourrait être utile dans un contexte où, par exemple, on voudrait analyser la régularité ou l'irrégularité du mouvement d'un objet.

Détection de cibles

Un module permettant d'identifier automatiquement des visages, des animaux, des voitures ou toute autre forme donnerait à l'algorithme une autonomie qui l'allègerait du besoin d'avoir un opérateur chargé de constamment sélectionner des cibles, des références à suivre. Il faut se rappeler qu'ultimement, l'algorithme devrait pouvoir fonctionner en temps réel et dans un contexte où les images seraient susceptibles de changer continuellement; un être humain serait beaucoup plus apte à interpréter les résultats de suivis d'objets suspects que lui fournirait un système qu'à sélectionner incessamment de nouvelles références.

Bien que l'oeil humain soit très habile à distinguer et reconnaître des formes familières, il existe des algorithmes simples de détection de cibles basés sur des banques d'images. Implémenter ces heuristiques serait opportuns dans un contexte où l'autonomie de l'algorithme est primordiale.

Suivi d'objets multiples

Enfin, une fonctionnalité qu'il serait intéressant d'implémenter est la possibilité de suivre plusieurs objets à la fois dans une même séquence d'images. Dans un contexte où il y aurait plusieurs objets en mouvement simultanément, par exemple, dans une vidéo de surveillance où un groupe de personnes circulent, cela se révèlerait très utile.

Il faudra en particulier savoir gérer les cas où des objets se croiseraient et où l'un d'eux serait occlus par l'autre. De façon similaire, les cas où un objet deviendrait deux objets distincts (par exemple, deux personnes marchant ensemble qui se séparent) devront également être pris en charge.

Conclusion

Une bonne compréhension du traitement numérique d'images se passe difficilement de l'étude et de l'implémentation d'algorithmes élémentaires. En effet, s'attaquer directement à un algorithme d'apparence aussi complexe et volumineux peut se révéler excessivement difficile. Il est également important de mentionner que de solides bases en mathématiques et en statistiques sont d'une très grande aide; dans le cas de la corrélation par filtre, par exemple, on ne saurait trouver la longueur d'un vecteur de plusieurs centaines de dimensions sans maîtriser certains concepts d'algèbre vectorielle.

Le but initial du projet étant d'implémenter l'algorithme Mean Shift et de rendre son fonctionnement à la fois efficace et robuste, l'objectif a été atteint. Pour ce faire, j'ai dû acquérir plusieurs connaissances et compétences en traitement d'image (voir Chapitre I de ce document). Le projet a également été pour moi une occasion de revisiter sur plusieurs notions de mathématiques qui ne m'avaient pas servi depuis longtemps (développement de Taylor, produit scalaire, opérations statistiques, etc.) Ce projet m'a fait réaliser que le traitement d'image est un domaine vaste qui n'a pas été pleinement exploré encore. De plus, c'est un domaine qui a un grand potentiel aux yeux de l'industrie et mon projet a été une très bonne première expérience.

L'objectif principal que je m'étais fixé, c'est-à-dire de tout simplement réussir l'implémentation de l'algorithme qui m'avait été présenté, a été rempli. Un objectif secondaire que je m'étais donné était de perfectionner mes compétences en optimisation de code; ayant moi-même vu l'évolution qu'a suivie mon code, je dois reconnaître avoir également atteint cet objectif. Finalement, la perspective globale que je retire de ce projet est le fait que toute bonne implémentation d'un algorithme dépend presque entièrement de la théorie qui se trouve derrière; sans l'article détaillant chaque étape de Mean Shift, je n'aurais jamais pu l'implémenter aussi rapidement.

Bibliographie

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

[2] Lipschutz, S. (1994). *Algèbre linéaire*. Maidenhead : McGraw-Hill.

[3] Sanders, D. H. & Allard, F. (1992). *Les statistiques : une approche nouvelle*. Montréal : Chenelière/McGraw-Hill.

[4] Comaniciu, D. Ramesh, V. & Meer, P. (2003). Kernel-Base Object Tracking. *IEEE Transactions on pattern analysis and machine intelligence*, *25*, 564-577.