Compte rendu

Projet : N°6 ESTIMATION DU MOUVEMENT

UE 3EE103 : Méthodes numériques en C

Chargé de TP : Thomas Dietenbeck

14/01/2022

Étudiants :
Axel Pham
Marc Zhan
Yousri Aboudaoud

Parcours : L3 EEA CMI ÉLECTRONIQUE

Groupe de TP : A1

INTRODUCTION

Le flux optique est la distribution des vitesses apparentes de mouvement des patterns de luminosité dans une image. Le flux optique peut provenir du mouvement relatif des objets et du spectateur [1]. Nous pouvons appliquer des algorithmes d'estimation du flot optique dans la compression vidéo ainsi que dans la robotique.

I. Estimation du flot optique par méthode de Horn & Schunck

1. La méthode de Horn & Schunck

La méthode de Horn-Schunck est une méthode d'estimation du flux optique basée sur le gradient. Elle est également connue sous le nom de méthode différentielle car elle utilise la fonction de gradient pour calculer le vecteur vitesse du pixel.

L'algorithme de Horn-Schunck ajoute une hypothèse de régularité global à l'équation de contrainte de base du flux optique. On suppose que le changement du flux optique est régulier sur l'ensemble de l'image, c'est-à-dire que le vecteur de mouvement de l'objet est régulier ou ne change que lentement

L'algorithme de Horn-Schunck appartient à la catégorie des flux optiques dense et, en raison de la simplicité, cette méthode a été largement utilisée et étudiée.

2. Estimation des dérivées partielles

Cette section présente l'estimation des dérivés partielles de l'intensité de l'image. L'on suppose l'intensité ne varie pas au cours du temps, ce qui peut être traduit comme suit :

$$I(p) = I(p + \Delta) \qquad (1)$$

Avec Δ le déplacement de l'objet ($\Delta=(\delta x,\delta y,\delta t)$). Si ce dernier se dernier se déplace faiblement l'on peut effectuer un développement de Taylor autour du pixel p=(x,y,t) comme qui suit :

$$I(p + \Delta) = I(p) + \delta x \frac{\partial I}{\partial x}(p) + \delta y \frac{\partial I}{\partial y}(p) + \\ + \delta t \frac{\partial I}{\partial t}(p) \qquad (2)$$

On remplace (1) dans (2), et on obtient l'équation de constance de la luminosité :

$$Ix(p)u + Iy(p)v + It(p) = 0 \qquad (3)$$

Avec (u,v,1)=$(\delta x/\delta t,\delta y/\delta t,1)$ le vecteur vitesse et (Ix,Iy,It) les gradients respectivement horizontal, vertical et temporel. L'on peut les déterminer en utilisant une méthode d'estimation numérique à savoir les différences finies :

$$f^{(1)}(t_0) = \frac{f(t_0+\Delta t) - f(t_0-\Delta t)}{2\Delta t} + O(\Delta t^2) \qquad (4)$$

En utilisant cette expression de différenciation centrée (pour n'importe quelle dimension x,y,t), on peut estimer la valeur de la dérivée d'un pixel. Le terme $O(\Delta t^2)$ est un terme d'erreur, ce qui nous pousse à utiliser une autre méthode. Barron et al. (1994) proposent une autre méthode plus précise, appliquer un filtre gaussien spatio-temporel (dont les coefficient sont [1/12 (-1; 8; 0; 8; 1)][1]. L'estimation du gradient s'effectue en utilisant 4 points de différenciation, étant donné que dans notre présent projet nous n'utilisons que deux images afin d'estimer le mouvement, nous utiliserons la formule (4) pour la dimension du temps.

D'autre part, il faut prendre en considération le fait qu'un pixel puisse se trouver au niveau des frontières d'une image (Pixel situé en dans un coin de l'image par exemple), pour résoudre cela nous utilisons le "padding" (rembourrage en français) avec lequel on ajoute un espace supplémentaire autour de l'image (Cf. Figure 1)

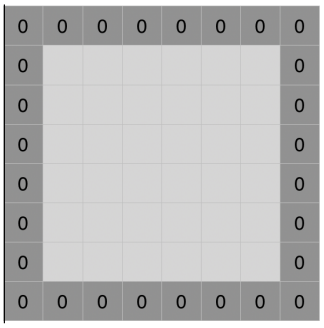


Figure 1. Rembourrage de zéro [2]

Voici ci-après un bout de code en langage C permettant l'application de ce qui vient d'être dit :

```
// Calcul de It
i_derive[2][i][j]  =  bmpImg_copy.img[i][j]  -
bmpImg.img[i][j] ;
// Calcul de Ix
if(j==0)
        i_derive[0][i][j]  =  (0*bmpImg.img[i][j]  -
8*bmpImg.img[i][j+1] + bmpImg.img[i][j+2]) / 12.
;
```

Figure 2. Langage C

Ce bout de code provient de la fonction calcul_intensite (de la bibliothèque Estimation.h).

3. Détermination des vitesses

Afin de résoudre (3) et déterminer le vecteur (u,v,1) de vitesse, l'on est obligé de rajouter des contraintes afin d'augmenter le nombre d'équations (pour le moment nous n'avons qu'une équation avec deux inconnus à déterminer u et v). L'équation (3) décrit la contrainte imposée sur la luminosité d'un pixel.

La méthode d'Horn Schunck suppose que le flot du mouvement est lisse (les points avoisinant un pixel dans une image ont le même mouvement) ce qui nous renvoie à l'équation suivante [1] :

$$E_s = \sum_{ij} \frac{1}{4} \left[(u_{ij} - u_{i+1j})^2 + (u_{ij} - u_{ij+1})^2 + \right.$$
$$\left. + (v_{ij} - v_{i+1j})^2 + (v_{ij} - v_{ij+1})^2 \right.$$

Avec E_s l'erreur de régularité (i.e mouvement lisse), u_{ij}/v_{ij} les vitesses d'un pixel (Cf. Figure 3), $\frac{1}{4}$ un terme de normalisation.

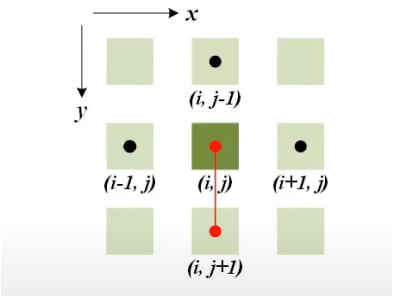


Figure 3. Pixel (i,j) d'une image[1]

Nous savons que le mouvement lisse signifie une variation entre les vitesses de chaque pixel nul, donc en reprenant l'expression de E_s, il faut que chaque terme de la somme soit nul (indépendance linéaire).

En prenant en compte les deux contraintes (luminosité d'un pixel constante, mouvement lisse)
Nous avons la fonction suivante :

$$E(\mathbf{V}) = \iint_{\Omega} (I_x u + I_y v + I_t)^2 + \alpha \left(\|\nabla u\|^2 + \|\nabla v\|^2 \right) d\mathbf{p}$$

Le second terme de la somme est E_s, avec α un poids (défini par l'utilisateur afin de contrôler l'importance d'une contrainte par rapport à l'autre). L'objectif désormais est de minimiser cette fonction quadratique (i.e Chercher un couple (u,v) pour lequel les dérivées partielles de E par rapport à u (respectivement v) soit nulles, cf. Figure 4).

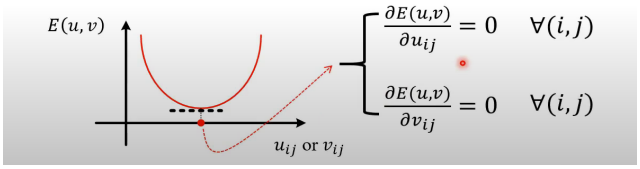


Figure 4. Condition pour minimiser E [1]

Après calcul des dérivées partielles l'on trouve que le couple (u,v) voulu est le couple des moyennes de (u,v) et donc l'on obtient les équations suivantes :

$$\begin{cases} \left(I_x^2 + \alpha\right) u + I_x I_y v & = \alpha\overline{u} - I_x I_t \\ I_x I_y u + \left(I_y^2 + \alpha\right) v & = \alpha\overline{v} - I_y I_t \end{cases}$$

Où $\overline{u}$ (resp. $\overline{v}$) est la moyenne de u (resp. v) autour du point **p.**

Après développement de ce système d'équations l'on trouve les expressions des vitesses :

$$u^{k+1} = \overline{u}^k - \frac{I_x\left[I_x\overline{u}^k + I_y\overline{v}^k + I_t\right]}{\alpha^2 + I_x^2 + I_y^2},$$

$$v^{k+1} = \overline{v}^k - \frac{I_y\left[I_x\overline{u}^k + I_y\overline{v}^k + I_t\right]}{\alpha^2 + I_x^2 + I_y^2}$$

La présence des exposants k+1 et k sera développée dans le chapitre suivant.

4. Estimation des vitesses d'écoulement

Pour estimer le mouvement apparu selon les pixels entre 2 images successives, on utilise les méthodes différentielles basées sur le flux optique. Pour commencer, on effectue la lecture des images pour pouvoir créer un tableau des matrices des images et prélever leurs données en pixels et en dimension. Puis, on calcule les dérivés partiels de l'intensité de l'image (*Ix*, *Iy*, *It*) et on crée deux matrices : une matrice pour des vecteurs vitesses u et v nulles, et une autre matrice pour la moyenne de ces vecteurs vitesse nulle.

```
// Lecture des images
BmpImg img1 =
readBmpImage("image1.bmp");
BmpImg img2 =
readBmpImage("image2.bmp");
// Lecture des dimensions de l'image
int dimX = img1.dimX, dimY = img1.dimY;

// Créer une matrice contenant les 2 images
float *** images = creerImage(img1,img2);
```

```
// Calcul des dérivées partielles (Ix, Iy, It)
float***
i_derive=calcul_intensite(images,img1);

// Créer une moyenne des vecteurs vitesse
nulle
float*** Moy_u_v= creerMatrice(2,dimX,dimY);

// Créer une matrice des vecteurs vitesse nulle
float*** u_v = creerMatrice(2,dimX,dimY);
```

Figure 5. Code de création de matrices

Maintenant, il nous revient à estimer le mouvement de l'image par une minimisation (calcul itératif des vecteurs vitesse u et v, et ainsi que sa moyenne) :

```
// Minimisation avec 100 itérations
int i=0;
while(i<100){
   u_v = calcul_vitesse(i_derive,Moy_u_v,0.3,
img1);

Moy_u_v=calcul_moyenne_vitesse(u_v,img1);
  i++;
}
```

Figure 6. Code d'estimation

Enfin, à la fin des calculs on obtient les résultats des vecteurs vitesses u et v. Mais on remarque que le temps utilisé pour traiter une grande image afin d'estimer le mouvement est lent. Par exemple, pour une image de 1336×1500 pixels, on doit utiliser d'environ 16 secondes pour traiter tous les algorithmes. En plus de cela, pour récupérer les données sur un fichier ".txt", il faut aussi du temps en plus. Par exemple, pour une image de 1336×1500 pixels, on doit utiliser environ 21 secondes au total pour traiter tous les algorithmes et récupérer les données.

5. Affichage sous Matlab

Puisque nous avons enregistré les matrices u et v chacune dans un fichier texte distinct (file et file2), nous allons récupérer les données (afin de les afficher sous matlab) via la commande readmatrix, nous récupérons aussi

les dimensions des matrices u et v afin de créer les axes X et Y. Finalement, nous affichons le champ de vecteur en utilisant la commande quiver.

II. Estimation rapide, méthode des pyramides

1. Méthode des pyramides

En traitement d'image, la pyramide est une représentation multi-résolution d'une image. Elle est souvent utilisée car elle permet non seulement d'accélérer l'algorithme de traitement d'image, mais aussi d'estimer des mouvements plus importants (taille en pixels) en travaillant avec des images sous-échantillonnées (moins de pixels).

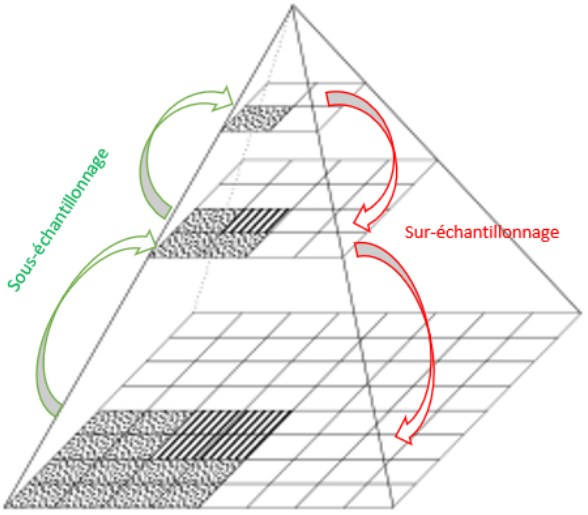


Fig. 1. On utilise un facteur 2 le long de chaque direction (sous et sur échantillonnage).

2. Sous-échantillonnage

Lorsqu'une image est importante en taille (en pixel), le temps utilisé pour traiter l'image afin d'estimer le mouvement devient important. Pour accélérer cet algorithme de traitement d'image, on utilise la méthode de multi-résolution qui réduit la taille d'une matrice par un sous-échantillonnage (voir Fig. 1-2-3) :

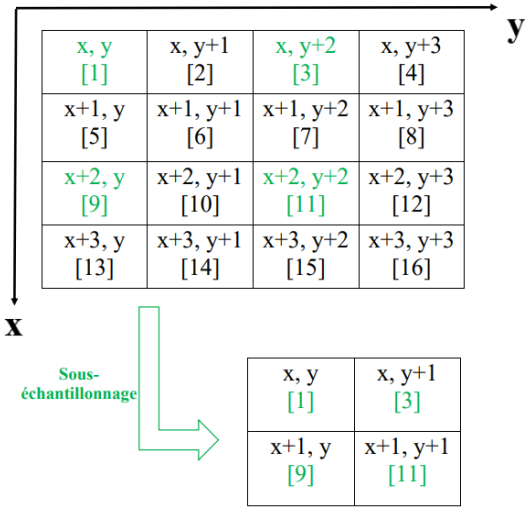


Fig. 2. Sous échantillonnage

Le sous-échantillonnage divise en 2 la taille de la matrice, dimensions 'x' et 'y'. Lorsque la dimension de 'x' ou de 'y' n'est pas paire (avant l'échantillonnage), alors la taille deviendra, après le sous-échantillonnage, le plus grand entier qui est inférieur à la taille 'x/2' pour 'x' ou 'y/2' pour 'y' (voir Fig. 2-3). En plus de cela, la matrice sous-échantillonnée récupère environ une valeur sur 4 et ces valeurs récupérées, elles sont entourées par des valeurs non récupérées (voir Fig. 3).

```
int dimX = img.dimX/2, dimY = img.dimY/2;
for(int i=0; i<dimX; i++){
    for(int j=0; j<dimY; j++){
        minimisation[0][i][j] = u_v[0][i*2][j*2];
        minimisation[1][i][j] = u_v[1][i*2][j*2];
    }
}
```

Fig. 3. Code de sous échantillonnages

3. Estimation des dérivés partielles et des vitesses d'écoulement

Pour accélérer l'algorithme de traitement d'image, on sous-échantillonne d'abord les deux images pour afin pratiquer les calculs moins lourds. A partir de ces images réduisent (en dimensions), on calcul ensuite les dérivés

partiels de l'intensité de l'image (*Ix*, *Iy*, *It*) et on crée deux matrices : une matrice pour des vecteurs vitesses u et v nulles, et une autre matrice pour la moyenne de ces vecteurs vitesse nulle (voir Fig. 4).

```
// Sous-échantillonnage des images
float***
minimisation=sous_echan(images,img1);
// Lecture des dimensions de ces images
int dimX1=img1.dimX/2, dimY1=img1.dimY/2;

// Calcul des dérivées partielles (Ix, Iy, It)
float*** i_derive_echan
=calcul_intensite_echant(minimisation,img1);

// Créer une moyenne des vecteurs vitesse
nulle
float*** Moy_u_v_echan
=creerMatrice(2,dimX1,dimY1);

// Créer une matrice des vecteurs vitesse nulle
float*** u_v_echan
=creerMatrice(2,dimX1,dimY1);
```

Fig. 4. Traitement sous-échantillonnage

Ensuite, il nous revient à estimer le mouvement de l'image par un minimisation (calcul itératif des vecteurs vitesse u et v, et ainsi que sa moyenne) (voir Fig. 5):

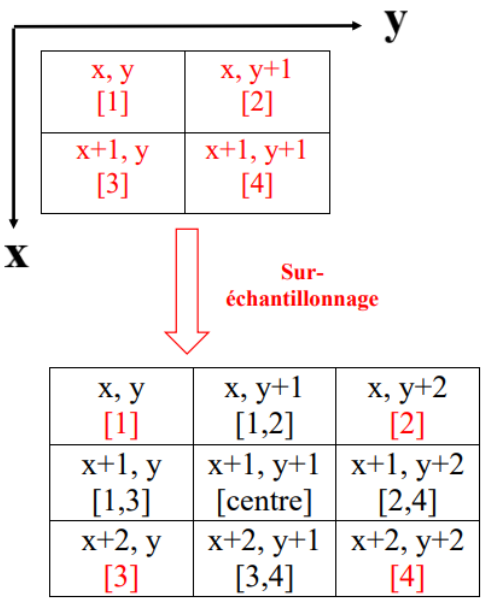
```
// Minimisation avec 100 itérations
int i = 0 ;
while( i < 100 ) {
      // Calcul de u et v échantillonnés
       u_v_echan = calcul_vitesse_echan(
i_derive_echan,
Moy_u_v_echan,ALPHA,img1);
      // Calcul sa moyenne échantillonnés
    Moy_u_v_echan =
calcul_moy_vitesse_echant(u_v_sur img1) ;
       i++;
} //ALPHA = [0.1 ; 0.5]
```

Fig. 5. Langage C, code pour l'estimation du mouvement.

Enfin, à la fin des calculs on obtient les réponses des vecteurs vitesses u et v échantillonnés. Il nous revient donc à les sur-échantillonnés pour obtenir une estimation de mouvement sur une image de taille approximativement égale à celle de son entrée.

4. Sur-échantillonnage

Lorsque l'on utilise la méthode de multi-résolution qui réduit la taille d'une matrice par un sous-échantillonnage, il nous faudra aussi un sur-échantillonnage pour augmenter la taille de la matrice à la fin des traitements de l'algorithme (voir Fig.6).



- *[a,b] = ([a] + [b]) / 2*
- *[centre] ≃ ([1,2] + [3,4]) / 2*
  ≃ ([1,3] + [2,4]) / 2

Fig. 6. Sur-échantillonnage

Le sur-échantillonnage augmente la taille de la matrice. Elle multiplie en 2 la taille de la matrice sous-échantillonnée puis on obtient sûrement des dimensions de 'x' et 'y' impaires (voir Fig. 6-7).

```
// dimX est la dimension en 'x' de l'image
// dimension en 'x' sous-échantillonné
int dimX1 = dimX/2;
// dimension en 'x' sur-échantillonné
int dimX2 = (dimX*2)-1;
// idem pour la dimension en 'y'
```

Fig. 7. Langage C, dimension d'une matrice sur-échantillonnée selon l'image.

5. Bilan (temps de calcul)

A la fin des calculs d'échantillonnage, on remarque qu'on obtient plus rapidement les résultats des vecteurs vitesses u et v. Par exemple, pour une image de 1336×1500 pixels, on utilise environ 4 secondes pour traiter tous les algorithmes. En plus de cela, pour cette même image, on utilise seulement 5 secondes au total pour traiter tous les algorithmes et récupérer les données.

III. Combinaison entre une méthode locale et globale

La méthode d'Horn-Schunck employées dans les deux premières parties est efficace, néanmoins elle manque en robustesse, c'est une méthode globale et donc s'applique à tout pixel d'une image. Par conséquent, l'on se trouve avec des vecteurs vitesses présentes dans des emplacements de l'image où le mouvement est censé être inexistant (Leur amplitude est faible) (Cf. Image X).

Une autre méthode, celle de Lucas-Kanade (méthode locale), est proposée afin de remédier à ce point faible. Le principe de la méthode de Lucas-Kanade est le suivant :

- On choisit une sous matrice (comportant des pixels) de notre image
- L'on considère que la vitesse est constante

Par conséquent, en prenant une sous matrice de taille 3*3, l'on se retrouve avec un système de 9 équations de constance de la luminosité [1]

$$\begin{bmatrix} f_{x1} & f_{y1} \\ \vdots & \vdots \\ & \\ & \\ f_{x9} & f_{y9} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -f_{t1} \\ \vdots \\ \\ \\ -f_{t9} \end{bmatrix} \quad (1)$$

Avec f la variation de la luminosité (I.E I dans les formules précédentes). Nous sommes amenés de nouveau (comme nous l'avons fait avec la méthode d'Horn-Schunck) à minimiser un terme de la forme

$$E_{LK} = \sum_i (f_{xi}u + f_{yi} + f_{ti})^2 \quad (2)$$

En procédant de la même manière qu'avec la manière d'Horn-Schunk nous allons calculer la dérivée de chacune des équations en fonction de u (respectivement v). Nous trouvons alors le système suivant

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum f_{xi}^2 & \sum f_{xi}f_{yi} \\ \sum f_{xi}f_{yi} & \sum f_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum f_{xi}f_{ti} \\ -\sum f_{yi}f_{ti} \end{bmatrix} \quad (3)$$

L'on peut aisément remarquer la force de la méthode de Lucas-Kanade, en effet, lorsque l'on a une plage de valeur qui ne change pas en fonction du temps, le couple (u,v) sera équivalent au vecteur nul, puisque la matrice 2*1 qui se trouve dans le second membre de l'équation (3) (comportant deux sommes dont les éléments sont des produits entre la variation de la luminosité suivant x(respectivement y) et la variation de la luminosité suivant le temps) est nul lorsqu'il n'y a pas de changement entre deux photos. Par contre, le point faible de la méthode de Lucas-Kanade est de considérer que les vitesses d'une plage de pixel sont identiques, ce qui n'est pas forcément le cas.

Nous avons vu la méthode d'Horn-Schunck qui nous offre un moyen de calcul global (pour chaque pixel), c'est donc une méthode avantageuse par sa densité, mais qui n'est pas robuste. De l'autre côté nous avons vu la méthode de Lucas-Kanade qui est plus robuste. Nous allons donc créer une méthode hybride qui regroupe les deux mondes [2]

Pour ce faire, nous allons d'abord appliquer un filtre gaussien à nos images (afin de supprimer les bruits faisant partie du spectre des hautes fréquences avec un masque gaussien qui est un

filtre passe-bas puisque sa transformée de Fourier est lui-même). La méthode de Lucas-Kanade devra minimiser la formule suivante [2]

$$E_{LK} = W * (I_x u + I_y v + I_t)^2 \quad (4)$$

Avec W le filtre gaussien appliqué à chaque pixel de l'image via le produit de convolution *. L'équation (4) est équivalente à

$$E_{LK} = V(W * (\nabla I \nabla I^t)V^t) \quad (5)$$

Avec V=(u,v,1)^t le vecteur vitesse, $\nabla I = (I_x, I_y, I_t)$

L'équivalence entre (4) et (5) se retrouve en effectuant des produits matriciels.

En combinant ce terme avec celui de la constance de la vitesse provenant de la deuxième contrainte de la méthode d'Horn-Schunk, l'on doit minimiser l'équation suivante

$$E(V) = \iint_{\Omega} V(W * (\nabla I \nabla I^t)V^t) \; ++ \; \alpha(\|\nabla u\|^2 + \|\nabla v\|^2)d \quad (6)$$

Minimiser (en passant par les équations d'Euler) cette dernière nous donne le système d'équations suivant :

$$\begin{cases} (W * I_x^2 + \alpha)\, u + W * (I_x I_y)\, v = & \alpha\bar{u} - W * (I_x I_t) \\ W * (I_x I_y)\, u + (W * I_y^2 + \alpha)\, v = & \alpha\bar{v} - W * (I_y I_t) \end{cases}$$

Les expressions des vitesses ressemblent à celles qui ont été trouvées dans le chapitre 3. Détermination des vitesses" à la seule différence qui est la présence du filtre.

Dans notre projet l'on implémente une fonction qui calcul le filtre gaussien de taille 3*3 dans la bibliothèque GaussianFilter.h. Nous appliquons ce filtre avec la fonction produit_Conv (dans la même bibliothèque) à nos images ; nous récupérons nos matrices d'images lissées. Nous effectuons le calcul de (u,v) par le biais d'une fonction CalcU_V_Conv (dans la même bibliothèque). Le résultat est illustré sur la figure 1(d)

Conclusion

Durant le projet nous avons fait face à un certain nombre de bugs dont :

- Des bugs lors de l'exécution du code dont l'adresse est 0xC0000005 ce qui signifie un dépassement des limites d'un tableau (nous avions des boucles où un des indices dépassait les bornes d'un tableau donné)
- Des bugs lors de l'exécution du type "Heap corruption" (corruption de tas en français) dû à une mauvaise utilisation de la mémoire alloué dynamiquement
- Des problèmes d'orientation lors de l'affichage sur Matlab (Exemple : le mouvement était orienté vers la gauche tandis que la femme tournait sa tête vers la droite)
- Des conflits de dimensions suite à un sous-échantillonnage de l'image

Les améliorations que l'on propose sont les suivantes :

- Implémentation d'une fonction qui ajuste les dimensions d'une image suite à un sous-échantillonnage
- Usage de structure (Variation de l'intensité, vecteurs vitesses…) avec des champs (Variation de l'intensité suivant l'axe X, matrice des vecteurs vitesses V…)

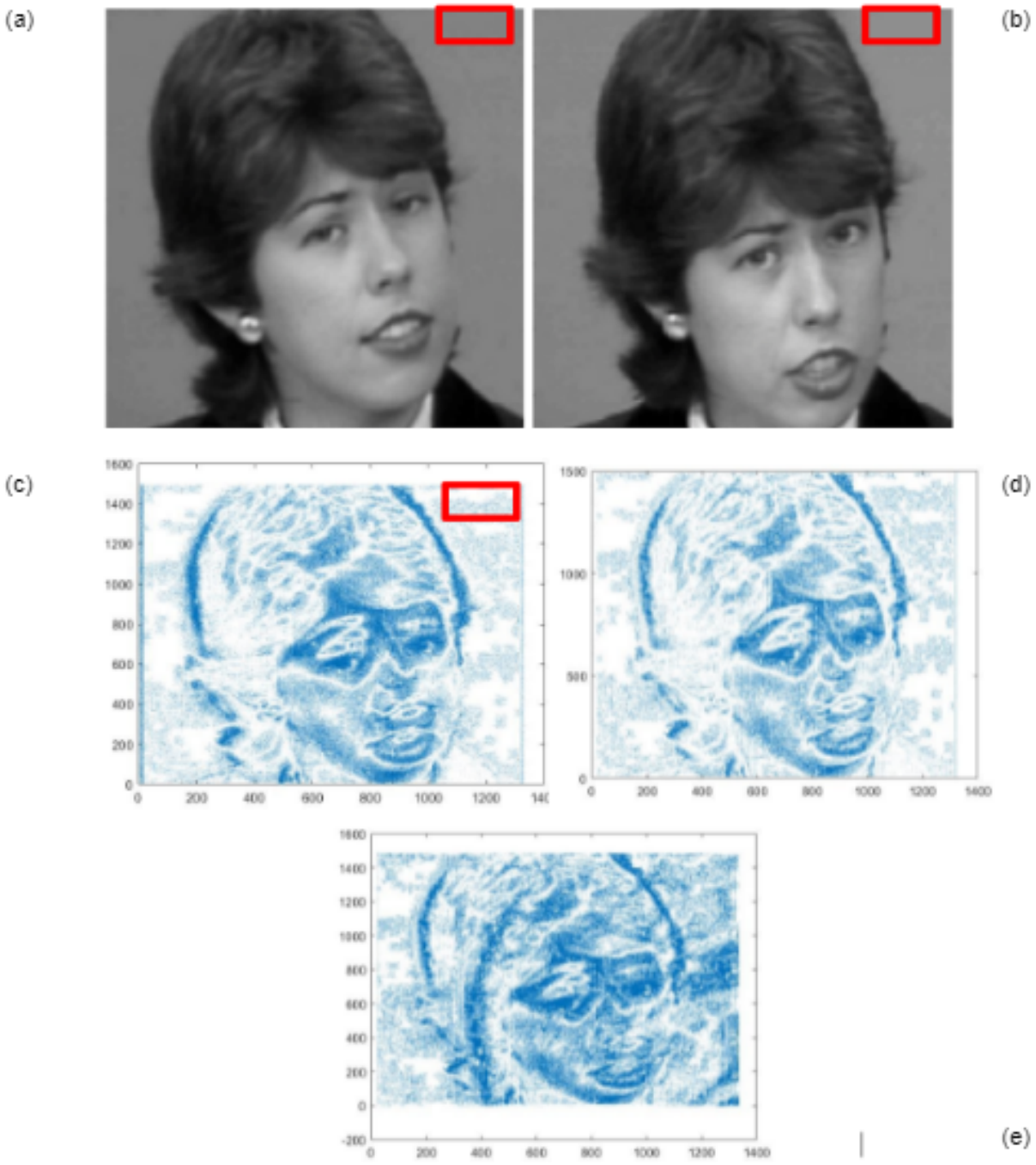


Figure 1. (a) et (b) deux images consécutives d'une personne qui déplace son visage (le cadre rouge est un emplacement de l'image où le mouvement est nul.
(c) et (d) Estimation de mouvement (en utilisant la méthode d'Horn-Schunk) entre les deux frames (I.E.images), le rectangle rouge contient des vecteurs de vitesses alors qu'ils devraient être nuls en considérant (a) et (b). (c) est le résultat de l'estimation avec la méthode des pyramides,(d) avec.
(e)Application de la combinaison des deux méthodes locale et globale.

Bibliographie

I. Estimation du flot optique par méthode de Horn & Schunck

2.Estimation des dérivées partielles

[1] Barron, J. L., et al. « Performance of Optical Flow Techniques ». *International Journal of Computer Vision*, vol. 12, nº 1, février 1994, p. 43-77. *DOI.org (Crossref)*, https://doi.org/10.1007/BF01420984.

[2]« CNN | Introduction to Padding ». GeeksforGeeks, 23 juillet 2019, https://www.geeksforgeeks.org/cnn-introduction-to-padding/.

3. Détermination des vitesses

[1]quarter DIP: Determining Optical Flow: Horn and Schunck Method. www.youtube.com, https://www.youtube.com/watch?v=HdPnPLxjJ9c. Consulté le 10 janvier 2022.

III. Combinaison entre une méthode locale et globale

[1]Lecture 7 - Optical Flow - 2014. *www.youtube.com*, https://www.youtube.com/watch?v=kJouUVZ0QqU. Consulté le 14 janvier 2022.

[2] A. Bruhn and J. Weickert and C. Schn ¨orrLucas/Kanade meets Horn/Schunck : Combining local and globaloptic flow methods(2005). Intl. J. of Computer Vision, vol 61 (3), pp.211–231