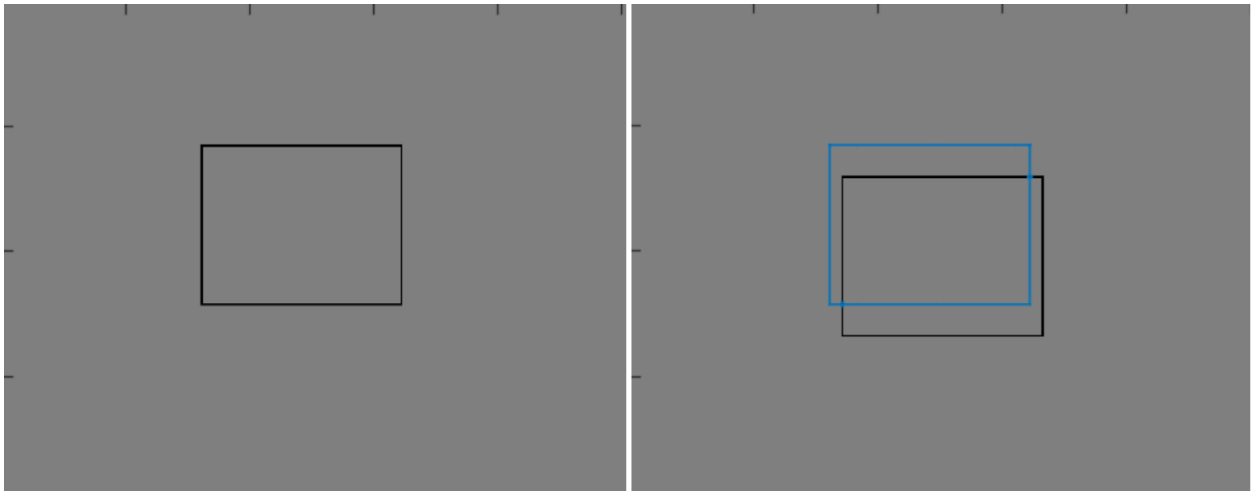
L'ESTIMATION DU MOUVEMENT

Programmation en Langage C



XIA Jieli
NAIT ACHOUR Younes
ZIANI Youcef
SAIDANI Nidhal

07/01/2022

LU3EE103 – Groupe C2

TABLES DES MATIÈRES

I – INTRODUCTION :

Le flux optique est un champ de déplacement visuel qui permet d'expliquer des variations dans une image animée en termes de déplacements de pixels. Le calcul de ce dernier permet d'estimer le déplacement des objets entre deux images successives.

Le projet réalisé a pour but d'estimer le mouvement, en utilisant la méthode de **Horn & Schunck** qui est une méthode différentielle basée sur le flux optique. Cette méthode suppose également que l'intensité lumineuse d'un objet ne varie pas sur l'image au cours du temps. Cependant, cette méthode présente quelques lacunes comme on a pu le voir durant les séances de TPs, tel qu'un temps d'estimation relativement long pour des images de taille importante, ainsi que des difficultés à estimer des mouvements à plus grande échelle, ce qui nous a amené à consolider cette dernière par une méthode de **Multi Résolution,** qui consiste à réduire la taille de l'image par sous échantillonnage**,** et par la méthode du **Filtrage de Gauss,** pour améliorer la robustesse de l'estimation**.**

II – PROBLÈME TRAITÉ :

❖ MÉTHODE de HORN & SCHUNCK :

La méthode de **Horn & Schunck** est basée sur le flux optique de l'image. Si nous considérons que l'intensité de la lumière ne varie pas sur un objet entre 2 images, nous pouvons alors estimer que chaque variation d'intensité est liée à un déplacement. Cela nous permet d'obtenir l'équation suivante avec I(p) la variation d'intensité lumineuse et p(x, y, t) le pixel en fonction des coordonnées de l'images et du temps :

$$I(p) = I(p + \Delta) \text{ avec } \Delta = (\delta x , \delta y , \delta t) \quad \ldots\ldots\ldots\ldots\ldots(1)$$

En faisant un développement de Taylor à l'équation (1) on obtient l'équation (2) :

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

En remplaçant (2) dans (1) on trouve l'équation (3) :

$$Ix(p)*u + Iy(p)*v + It(p) = 0 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

Avec : $\frac{\partial I}{\partial x} = Ix$, $\frac{\partial I}{\partial y} = Iy$ et $\frac{\partial I}{\partial t} = It$ les gradients horizontal (suivant x), vertical (suivant y) et temporel (suivant t). Et $\frac{\partial x}{\partial t} = u$, $\frac{\partial y}{\partial t} = v$ et $\frac{\partial t}{\partial t} = 1$ les composantes du vecteur vitesse V = (u,v,1)

L'équation (3) étant une équation à deux inconnus, sa résolution nécessite l'ajout de contraintes. Pour répondre à ce problème, la méthode de Horn & Schunck suppose que le champ de mouvement est lisse ce qui revient à minimiser l'énergie [1] :

$$E(V) = \iint_{\Omega} [Ix(p)u + Iy(p)v + It(p)]^2 + \alpha(\|u\|^2 + \|v\|^2)dp \quad \ldots..(4)$$

On cherche donc à trouver V qui minimise E, ce qui revient à annuler les dérivées premières de E par rapport à u et v :

$$\frac{\partial E(V)}{\partial u} = 0 \qquad et \qquad \frac{\partial E(V)}{\partial v} = 0$$

Ce qui nous permet d'obtenir les équations suivantes :

$$2\,Ix.(Ix.u + Iy.v + It) + 2\alpha \|u\|^2 = 0 \qquad et \qquad 2\,Iy.(Ix.u + Iy.v + It) + 2\alpha\|v\|^2 = 0$$

$$\rightarrow Ix^2.u + Ix.Iy.v + Ix.It + \alpha\|u\|^2 = 0 \qquad et \qquad Ix.Iy.u + Iy^2.v + Ix.It + \alpha\|v\|^2 = 0$$

En posant l'approximation du Laplacien suivante : $\|u\|^2 \approx \bar{u} - u$ et $\|v\|^2 \approx \bar{v} - v$, on retrouve le système d'équations (10) du descriptif fourni sur moodle[2].

$$\rightarrow \{(Ix^2 + \alpha).u + Ix.Iy.v = \alpha.\bar{u} - Ix.It \quad Ix.Iy.u + (Iy^2 + \alpha).v = \alpha.\bar{v} - Iy.It$$

- <u>Résolution du système d'équations</u> : On met le système sous forme matricielle :

$$\begin{pmatrix} I_x^2+\alpha & I_xI_y \\ I_xI_y & I_y^2+\alpha \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \alpha\bar{u} - I_xI_t \\ \alpha\bar{v} - I_yI_t \end{pmatrix} \quad avec\ M1 = \begin{pmatrix} I_x^2+\alpha & I_xI_y \\ I_xI_y & I_y^2+\alpha \end{pmatrix}$$

  Det (M1) = $(Ix^2 + \alpha).(Iy^2 + \alpha) - (Ix.Iy).(Ix.Iy) = \alpha.(\alpha + Ix^2 + Iy^2)$

- <u>Calcul de u</u> :

$$u = \frac{\begin{vmatrix} \alpha\bar{u}-Ix.It & Ix.Iy \\ \alpha\bar{v}-Iy.It & Iy^2+\alpha \end{vmatrix}}{\alpha.(\alpha+Ix^2+Iy^2)} = \frac{(\alpha\bar{u}-Ix.It).(Iy^2+\alpha)-(Ix.Iy).(\alpha\bar{v}-Iy.It)}{\alpha.(\alpha+Ix^2+Iy^2)} = \frac{(\alpha+Ix^2+Iy^2).\bar{u}-(Ix.\bar{u}+Iy.\bar{v}+It).Ix}{(\alpha+Ix^2+Iy^2)}$$

$$\rightarrow u = \bar{u} - \frac{(Ix.\bar{u}+Iy.\bar{v}+It).Ix}{(\alpha+Ix^2+Iy^2)} \quad \text{.....(5)}$$

- <u>Calcul de v</u> :

$$v = \frac{\begin{vmatrix} \alpha\bar{u}-Ix.It & Ix.Iy \\ \alpha\bar{v}-Iy.It & Ix^2+\alpha \end{vmatrix}}{\alpha.(\alpha+Ix^2+Iy^2)} = \frac{(\alpha\bar{u}-Ix.It).(Ix^2+\alpha)-(Ix.Iy).(\alpha\bar{v}-Iy.It)}{\alpha.(\alpha+Ix^2+Iy^2)} = \frac{(\alpha+Ix^2+Iy^2).\bar{v}-(Ix.\bar{u}+Iy.\bar{v}+It).Iy}{(\alpha+Ix^2+Iy^2)}$$

$$\rightarrow v = \bar{v} - \frac{(Ix.\bar{u}+Iy.\bar{v}+It).Iy}{(\alpha+Ix^2+Iy^2)} \quad \text{.....(6)}$$

- <u>Estimation de u et v de manière itérative</u> :

  Pour estimer u et v de manière itérative, on suppose par récurrence que :

$$\begin{cases} u = u^{i+1} \\ v = v^{i+1} \end{cases} et \begin{cases} \bar{u} = u^i \\ \bar{v} = v^i \end{cases}$$

On remplaçant dans les équations (5) et (6) on trouve les formules itératives de u et v :

$$\begin{cases} u^{i+1} = u^i - \dfrac{\left(I_x u^i + I_y v^i + I_t\right) I_x}{\left(\alpha + I_x^2 + I_y^2\right)} \\ v^{i+1} = v^i - \dfrac{\left(I_x u^i + I_y v^i + I_t\right) I_y}{\left(\alpha + I_x^2 + I_y^2\right)} \end{cases}$$

❖ MÉTHODE DE MULTI RÉSOLUTION :

Pour des images de grande taille, le calcul des vitesses par la méthode de Horn & Schunck prend un temps considérable. Il n'est pas tout à fait exact pour estimer des déplacements plus importants.

Pour résoudre ce problème, une méthode de sous-échantillonnage des images est proposée. En effet, on réduit la taille de l'image par un facteur 4 avec un système pyramidal. Puis on calcul sur les plus petites images, les vitesses u et v par la méthode précédente. Pour finir on interpole ces vitesses pour revenir à la taille initiale de l'image. On gagne un temps de calcul considérable et on résout le problème des grands déplacements au vue de la réduction de ces dernières par cette méthode.

❖ MÉTHODE DE LISSAGE DE L'IMAGE PAR UN FILTRE GAUSSIEN :

Dans l'optique d'améliorer la robustesse de l'estimation du mouvement, on combine les approches locales et globale traitées précédemment, puis on minimise l'équation (7) :

$$(V) = \iint_\Omega V^T\left(W * \left(\nabla I \nabla I^T\right)\right)V + \alpha(\|\nabla u\|^2 + \|\nabla v\|^2)dp \quad \text{.....(7)}$$

Où W est un filtre gaussien, et (*) l'opérateur de convolution. Le produit de convolution avec le filtre gaussien a pour but d'obtenir des images plus lisses, et de ce fait avec moins de bruits et avec des contours plus en évidence. L'expression de la gaussienne à 2 dimensions est donné par : $G(x,y) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$ et le filtre W est une matrice où la gaussienne est calculée à chaque point de celui-ci.

La minimisation de l'équation (7) se fait par le système suivant :

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

Avec la même méthode de résolution que pour Horn & Schunck, on obtient les formules itératives de u et v :

$$\begin{cases} u^{i+1} = u^i - \dfrac{W*(I_x^2 u^i) + W*(I_x I_y v^i) + W*(I_x I_t)}{(\alpha + W*I_x^2 + W*I_y^2)} \\ v^{i+1} = v^i - \dfrac{W*(I_y^2 v^i) + W*(I_x I_y u^i) + W*(I_y I_t)}{(\alpha + W*I_x^2 + W*I_y^2)} \end{cases}$$

III – DESCRIPTION DES MÉTHODES NUMÉRIQUES UTILISÉES :

1) Utilisation de dérivées premières (avant, arrière et centrée) :

Pour implémenter la méthode de Horn & Schunck, on a calculé les gradients directionnels et temporels à partir des images transformés en matrices 2D(avec leur niveau de gris). On utilise donc les équations différentielles issues du développement de Taylor :

$$f(t + \Delta t) = f(t) + \Delta t.\, f'(t) + \frac{\Delta t}{2}.\, f''(t) + reste$$

$$f(t - \Delta t) = f(t) - \Delta t.\, f'(t) + \frac{\Delta t}{2}.\, f''(t) + reste$$

$\rightarrow f'(t) = \frac{f(t+\Delta t)-f(t)}{\Delta t}$ Dérivée avant utilisée pour calculer les Ix sur les bords gauches de la matrice, et les Iy sur les bords du haut.

$\rightarrow f'(t) = \frac{f(t)-f(t-\Delta t)}{\Delta t}$ Dérivée arrière utilisée pour calculer les Ix sur les bords droits de la matrice, et les Iy sur les bords du bas.

$\rightarrow f'(t) = \frac{f(t)-f(t-\Delta t)}{\Delta t}$ Dérivée centrée utilisée pour calculer les Ix et Iy des pixels qui ne sont pas sur les bords.

2) Interpolation bilinéaire :

Pour que les vitesses calculées sur les images échantillonnées correspondent aux images d'origine, on est obligé de passer par une interpolation bilinéaire (entre deux pixels ou entre quatres pixels selon l'endroit que nous parcourons dans notre matrice). Les équations utilisées pour interpoler sont données comme suit :

- Pour les lignes pairs et colonnes impairs :

$f(x) = [(x_{right} - x)/(x_{left} - x_{right})].f(x_{left}) + [(x_{left} - x)/(x_{left} - x_{right})].f(x_{right})$

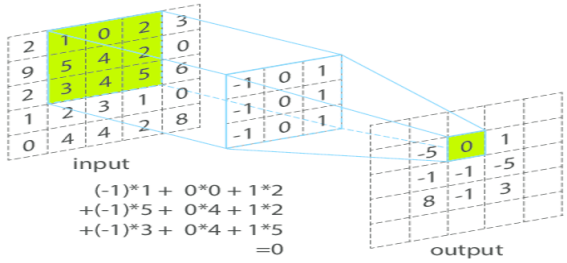
- Pour les lignes impairs et colonnes pairs :

$f(x) = [(x_{down} - x)/(x_{up} - x_{down})].f(x_{up}) + [(x_{up} - x)/(x_{up} - x_{down})].f(x_{down})$

- Pour les cases se situant entre 4 pixels : On fait la somme des pixels autour de cette case et on divise par 4 (résolution à voir dans la partie algorithmique du rapport).

3) Convolution :

Pour convoluer une matrice avec un filtre, on utilise une méthode de convolution terme par terme. On va calculer le nouveau terme avec le filtre en le tournant à 180°, puis en faisant une somme de produit terme à terme de la taille du filtre. Un exemple avec cette figure ci dessous :



[3]

IV – PRINCIPE DE L'ALGORITHME PROGRAMME ET STRUCTURATION DES DONNÉES :

❖ **BIBLIOTHÈQUE POUR LA MÉTHODE de HORN & SCHUNCK** :

Les bibliothèques utilisées pour implémenter cette méthode sont :

- **Horn_And_Schunk.h :** Contient des structures de gradients et de vitesses. Dans cette bibliothèque on a pu implémenter toutes les fonctions qui vont servir à estimer le mouvement par le méthode d'Horn & Schunck. Ces fonctions sont les suivantes :

  - **Calcul_U_V_Bar :** permet de calculer les vitesses moyennes $\overline{U}$ et $\overline{V}$. Au départ la matrice qui contient les vitesses moyennes est initialisée à zéro, puis actualisée à chaque nouvelle itération de U et V. L'algorithme qui permet de remplir cette matrice est donné par :

    Pour i de 0 à longueur-1 :
    Pour j de 0 à largeur-1 :
    $\overline{U} = \frac{U[i-1,j-1]+U[i-1,j]+U[i-1,j+1]+U[i,j-1]+U[i,j+1]+U[i+1,j-1]+U[i+1,j]+U[i+1,j+1]}{8}$
    Fin pour
    Fin pour

    Cependant cet algorithme n'est pas applicable en tous points de la matrice, car sur les bords de matrice on ne peut pas calculer la moyenne des 8 pixels autour, ce qui nous oblige à rajouter plusieurs cas particulier dans notre code C.

  - **Calcul_Gradient :** le calcul du gradient se fait par rapport aux axes x et y et par rapport au temps. L'algorithme est donné par :

    Pour i de 0 à longueur-1
    Pour j de 0 à largeur-1

$$\begin{cases} Ix = pixel[i,1] - pixel[i,0], & si\ j = 0 \\ Ix = pixel[i, largeur-1] - pixel[i, largeur-2], & si\ j = largeur-1 \\ Ix = \frac{pixel[i,j+1] - pixel[i,j-1]}{2} & \text{sinon} \end{cases}$$

$$\begin{cases} Iy = pixel[1,j] - pixel[0,j] & si\ \ i = 0 \\ Iy = pixel[longueur-1,j] - pixel[longueur-2,j] & si\ i = longueur-1 \\ Iy = \frac{pixel[i+1,j] - pixel[i-1,j]}{2} & \text{sinon} \end{cases}$$

    It = pixel_image2[i,j] – pixel_image1[i,j]
    Fin Pour
    Fin Pour

- **Calcul_Vitesses_UV :** cette fonction sert à calculer les vitesses de déplacement des pixels grâce aux équations (5) et (6) trouvées précédemment

❖ BIBLIOTHÈQUE POUR LA MULTI RÉSOLUTION :

- **Echantillonnage_Interpolation.h :** Cette bibliothèque contient les fonctions nécessaires au sous-échantillonnage des images et à l'interpolation du mouvement. Les fonctions sont décrites ci-dessous.

  - **Sous_Echantillonnage :** Réduit la taille de l'image en gardant une ligne sur deux de la matrice image, et en gardant un pixel sur deux pour les colonnes. Cette fonction permet d'avoir une image réduite de taille par un facteur 2. Il faudra donc l'utiliser deux fois pour avoir une pyramide d'image ou l'image la plus petite est réduite d'un facteur 4 par rapport à la taille initiale.

  - **Sur_Echantillonnage :** Cette fonction sert à interpoler les vitesses calculées par la méthode de Horn & Schunck sur la plus petite image. Son algorithme est donné par :
    Pour i de 0 à n :
    Pour j de 0 à m :

$$\begin{cases} Vitesse[i,j] = \dfrac{Vitesse[i,j+1] + Vitesse[i,j-1]}{2} & Pour\ les\ lignes\ pairs\ et\ colonnes\ impairs \\ Vitesse[i,j] = \dfrac{Vitesse[i+1,j] + Vitesse[i-1,j]}{2} & Pour\ les\ lignes\ impairs\ et\ colonnes\ pairs \\ Vitesse[i,j] = \dfrac{Vitesse[i-1,j-1] + Vitesse[i-1,j+1] + Vitesse[i+1,j+1] + Vitesse[i+1,j-1]}{4} & \text{sin}on \end{cases}$$

    Fin Pour
    Fin Pour
    Ceci ne s'agit que d'un algorithme d'interpolation général auquel on ajoute plusieurs cas particuliers dans le Code C.

❖ BIBLIOTHÈQUE POUR FILTRE DE GAUSS :

- **FiltreGaussien.h :** Elle contient les fonctions permettant de filtrer une matrice par convolution et le calcul des vitesses avec les gradients convolué par ce filtre.

  - **prod_mat_termeaterme :** Elle permet de faire le calcul des matrices termes à termes.

  - **filtre_gauss :** Cette fonction va calculer le filtre gaussien selon la taille voulu par l'utilisateur (détermination de sigma) sachant que le filtre gaussien est définie dans un intervalle de [-3*sigma;3*sigma]. Plus la taille est grande, plus l'image sera floue. Elle utilise une fonction **gaussienne** qui calcul la gaussienne en 2D selon les coordonnées x et y de la matrice du filtre.

  - **convolution :** Pour réaliser cette fonction on utilise d'autre fonction tel que **b** et **conv1terme**. On utilise plusieurs fonctions pour plus de lisibilité. La fonction b permet

de traiter les cas où l'on parcourt les bords de la matrice et de retourner 0 si elle est atteinte. Pour conv1terme, elle fait le calcul de la convolution avec le filtre pour 1 pixels soit une valeur de la matrice. Pour cela on utilise 2 boucles for en commençant par la fin car le filtre doit être tourné à 180° et on parcours le filtre de [-3*sigma;3*sigma] par rapport au pixel courant. La fonction convolution va ensuite utiliser conv1terme pour faire le calcul pour la matrice entière.

❖ **AUTRES BIBLIOTHÈQUES ET STRUCTURES:**

- **Operation_matrice.h :** Cette bibliothèque contient l'ensemble des fonctions permettant de créer, de lire et d'écrire une matrice dans un fichier texte, et de libérer l'espace alloué à une matrice.

- **INTERFACE.h :** contient les fonctions nécessaires à l'affichage et au fonctionnement du programme. Ils auraient pu être implémentés directement dans le main, mais par souci de clarté, on a préféré les séparer.

- Structure **Vitesses** permet de rassembler les vitesses u et v car ils doivent être calculés ensemble sinon la méthode d'estimation du mouvement ne marcherait pas.

- Structure **Gradient** part du même principe que Vitesses. On met les gradient Ix, Iy, It sous une même structure car en calculer un seul n'aurait aucun sens dans notre cas.

V – VALIDATION DE CHAQUE MÉTHODE :

Pour vérifier que nos 3 méthodes sont correctes on les a testés sur Matlab qui permet d'afficher les vecteurs des vitesses. Voici les résultats obtenus :

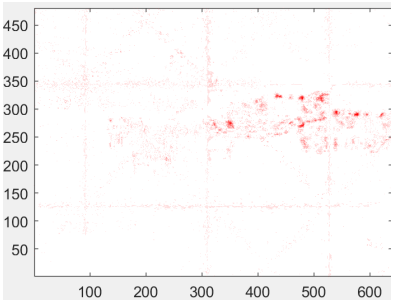
Image d'origine

Méthode de Horn & Schunck

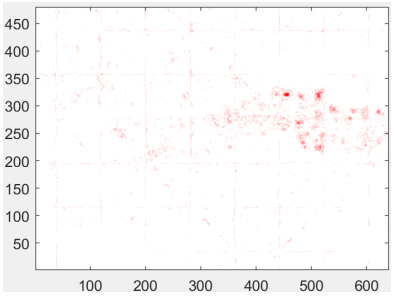
Méthodes Multi-résolution

Méthode avec le filtre gaussien

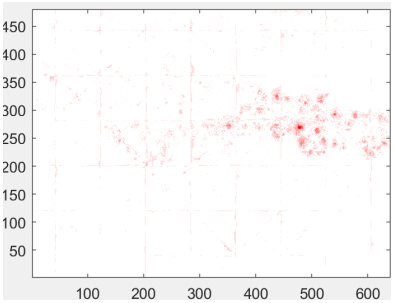
Le déplacement est bien calculé, il y a une forte concentration de flèches pour les plus grands mouvements. Pour voir plus en détaille on affiche que les vecteurs :



Horn & Schunck

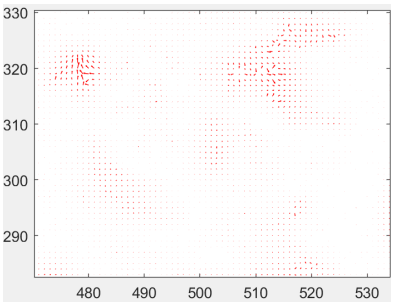


Multi Résolution

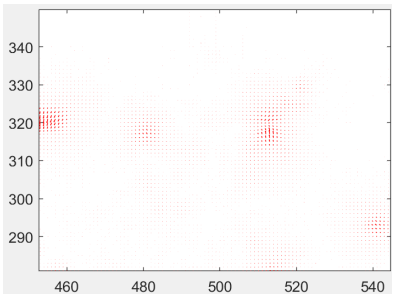


Filtre de Gausse

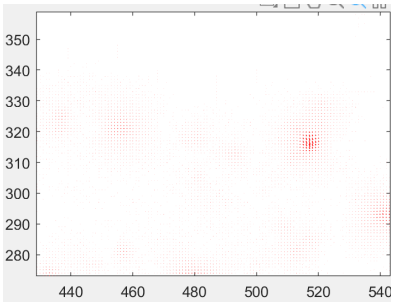
En Zoomant sur une même partie de l'image :



Horn & Schunck



Multi Résolution



Filtre de Gausse

On observe de légères différences entre chaque méthode. On voit qu'avec la multi résolution les flèches sont moins importantes, elle supprime les mouvements les plus légers. La troisième méthode qui est la combinaison des 2 autres avec le filtre gaussien permet de lisser le mouvement. La direction des flèches est moins abrupte et plus dense. Toutes nos méthodes marchent.

VI – CONCLUSION :

On peut voir avec les vecteurs vitesses vu ci dessus que notre code n'est pas optimal. En effet, un problème de bord est visible. La vitesse est bien calculée cependant elle comprend beaucoup de petits déplacements non nécessaires. Notre plus grand problème connu était justement de prendre en compte la limite de l'image, on a pas réussi à supprimer tous les effets des bords. Les conditions que nous avons imposées ne suffisent pas. Un autre problème rencontré était l'utilisation de "double". On avait au départ utilisé seulement des données de type "double" lorsque cela était nécessaire, cependant les données que l'on obtient dans le fichier texte n'étaient pas celles attendues. On a alors remplacé tous les types "double" par les types "float" et notre code calculait alors bien le mouvement.

On pourrait améliorer notre projet en ajoutant par exemple un seuil qui pourrait supprimer tous les faibles déplacements et mettre en valeur les grands déplacements en l'amplifiant. Notre programme fait actuellement les calculs pixels par pixels ce qui implique un très long temps de calcul dès que l'image est plus grande. Une amélioration possible serait d'optimiser ce temps de calcul.

La méthode de Horn & Schunck est une méthode qui peut estimer du mouvement cependant à faible échelle. Il a beaucoup de contraintes notamment que l'intensité lumineuse de chaque objet ne doit pas varier ce qui entraîne beaucoup de données indésirable que l'on pourrait négliger lors du calcul.

RÉFÉRENCE :

[1] Horn, Berthold K. P., et Brian G. Schunck. « Determining Optical Flow ». *Artificial Intelligence* 17, nº 1 (1 août 1981): 185‑203. https://doi.org/10.1016/0004-3702(81)90024-2.

[2]https://moodle-sciences.upmc.fr/moodle-2021/pluginfile.php/90847/mod_resource/content/10/Descriptif.pdf

[3]https://www.google.com/url?sa=i&url=https%3A%2F%2Fperso.esiee.fr%2F~perretb%2FI5FM%2FTAI%2Fconvolution%2Findex.html&psig=AOvVaw1crfKdKNhdzzW6NvDdNf1s&ust=1642261798223000&source=images&cd=vfe&ved=0CAsQjRxqFwoTCOiG9JvMsfUCFQAAAAAdAAAAABAD