

Chapitre 3. Mouvement (2 Semaines)

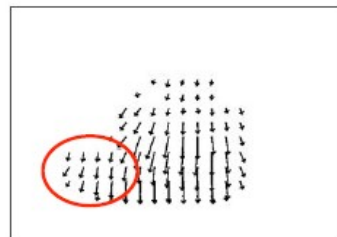
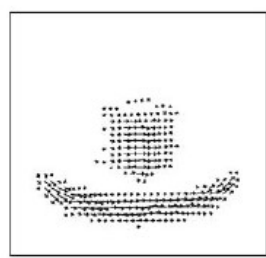
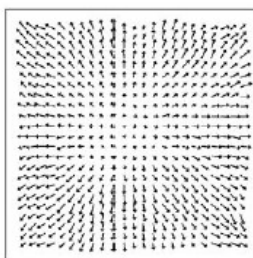
Estimation du mouvement et flot optique. Les algorithmes d'estimation de mouvement.
Reconstruction de structure en utilisant le mouvement.

Flot Optique

La mesure du flot optique est une étape de traitement de l'image dite de bas niveau. On lui trouve de nombreuses applications comme l'analyse de mouvements de fluides en physique expérimentale, la compression de séquences d'images vidéo par compensation de mouvement, ou son utilisation pour des phases de traitement des images de plus haute niveau, comme la reconstruction de scènes tridimensionnelles. Le terme de flot optique a été inventé par le psychologue James Jerome Gibson dans une étude sur la vision humaine. En 1980, Horn et Schunck [HS80, HS81] ont proposé une méthode d'estimation du flot optique basée sur la régularisation. Ce premier travail a été suivi d'un grand nombre de contributions qui ont proposé différentes méthodes alternatives. Pour n'en citer que quelques unes, nous pouvons parler des méthodes de filtrage spatiotemporel, basées sur les travaux de Adelson et Bergen [AB85], qui sont divisées en deux branches suivant que l'énergie [Hee88] ou la phase [FJ90] de la sortie des filtres est utilisée pour effectuer l'estimation. On peut également utiliser les techniques d'appariement par blocs [BA83, BYX83, Ana89]. En 1994, Barron, Fleet et Beauchemin [BFB94] ont fait un inventaire exhaustif des méthodes existantes.



Exemples:



Interprétation du flux

Calcul du mouvement apparent

(1) Le calcul d'un mouvement apparent *global* (mise en correspondance) entre deux images correspond à l'estimation des paramètres d'une transformation affectant *tous* les points de l'image : translation, rotation, homothétie, affinité,...

(2) Le calcul du mouvement apparent *local* consiste à associer à chaque pixel (x,y,t) de I un vecteur (v_x^t, v_y^t) représentant la *vitesse apparente* du pixel (x,y) à l'instant t .

—————→ Calcul du *flot optique* (= Champ de mouvement apparent)

Idéalement : le vecteur (v_x^t, v_y^t) représente la projection sur le plan image du vecteur vitesse (V_x^t, V_y^t, V_z^t) des objets de la scène par rapport au repère image (O,x,y,z) (grandeur objective).

On le calcule à partir des variations temporelles de la fonction $I(x,y,t)$.

Caen ETASW 2006
Archer MANZANERA - ENSTARIE
2

Modèle mathématique

Modèle de projection

Calcul du flux optique

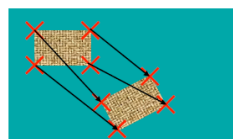
Deux approches

Il existe deux approches distinctes:

- Flux optique dense: son objectif : calculer le mouvement apparent (vitesse instantanée) de chaque pixel. Elle se base sur la continuité temporelle
- Mise en correspondance: son objectif : apparier certaines structures spatiales pour chaque couple d'images. Elle se base sur la discontinuité temporelle.



Flux optique dense



Mise en correspondance

Mise en correspondance (rappel)

Détection de points d'intérêt

- Éléments 1D : contours
- Éléments ponctuels : points d'intérêt, coins, Harris
- Éléments invariants / échelle : SIFT, SURF, MSER ...

Caractérisation de points

- Invariant translation : Patch luminance
- Invariant rotation : Histogrammes de gradient

Calcul du flux optique dense

Mouvement \Leftrightarrow *Variation de flux*

– Hypothèse d'éclairement constant

Estimation Différentielle Projetée du Flot Optique

Une séquence d'images est une fonction réelle $I(t; x_1, x_2)$ de trois variables t, x_1, x_2 que nous supposons pour l'instant continues. Nous utiliserons les notations abrégées x pour (x_1, x_2) et $x(t)$ pour $(x_1(t), x_2(t))$. Le modèle mathématique standard utilisé pour trouver des équations qui définissent le flot optique est basé sur **une hypothèse d'illumination constante** :

un point réel $[X_1(t) X_2(t) X_3(t)]$ de la scène est projeté sur le plan image de la caméra au point $(x_1(t), x_2(t))$

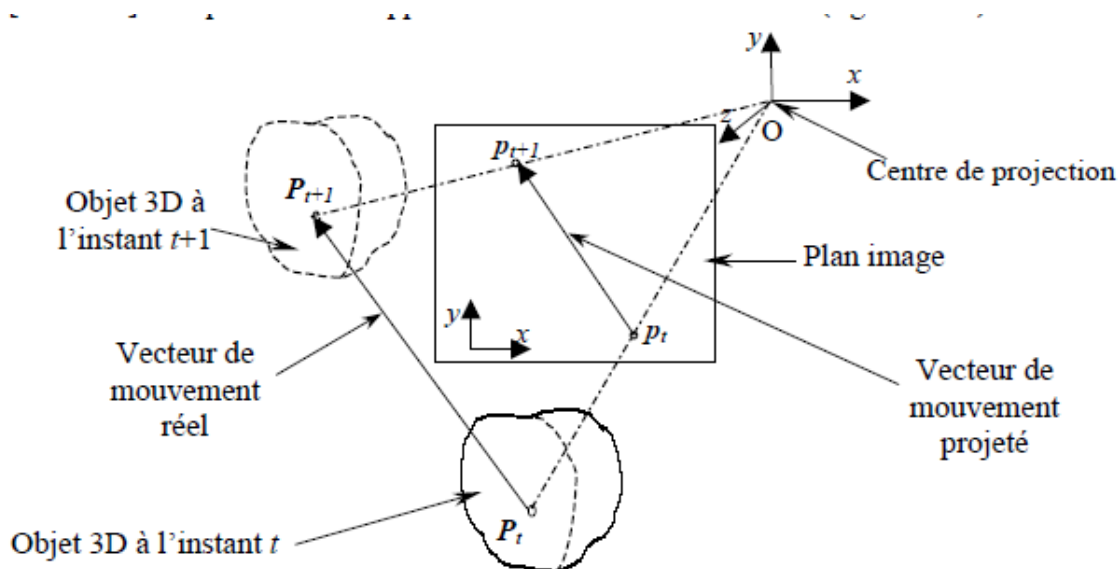


Illustration des mouvements réel et apparent, dans un système optique de prise de vues.

$$[X_1(t) \ X_2(t) \ X_3(t)] \rightarrow (x_1(t), x_2(t)) \quad \text{au temps } t$$

Le flot optique au temps t et au point $x(t)$ est alors défini comme la vitesse du point image

$$v = (v_1, v_2) = \left(\frac{dx_1}{dt}, \frac{dx_2}{dt} \right)$$

L'hypothèse d'illumination constante consiste à dire que la luminosité apparente du point

$(x_1(t), x_2(t))$ au temps t ne dépend pas du temps t ce qui écrit

$$I(t; x(t)) = I_0$$

Le flot optique est donc contraint par l'équation suivante

$$\frac{\partial I}{\partial t} + \nabla I \cdot \frac{dx}{dt} = 0$$

ou encore:

$$v \cdot \nabla I + \frac{\partial I}{\partial t} = 0$$

Nous introduisons dès à présent une variante de (OF) qui prend en compte des variations

d'illumination. On utilise un modèle de luminosité apparente lambertien.

$$I(t; x_1, x_2) = R(t; x_1, x_2) \times L(t; x_1, x_2)$$

Contrainte de luminosité constante

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$$

Formule de Taylor

$$= I(x, y, t) + \frac{\partial I}{\partial x} \cdot \delta x + \frac{\partial I}{\partial y} \cdot \delta y + \frac{\partial I}{\partial t} \cdot \delta t + \dots$$

Termes d'ordres
supérieures
négligés

SOIT :

$$\boxed{\nabla I \cdot v + \frac{\partial I}{\partial t} = 0}$$

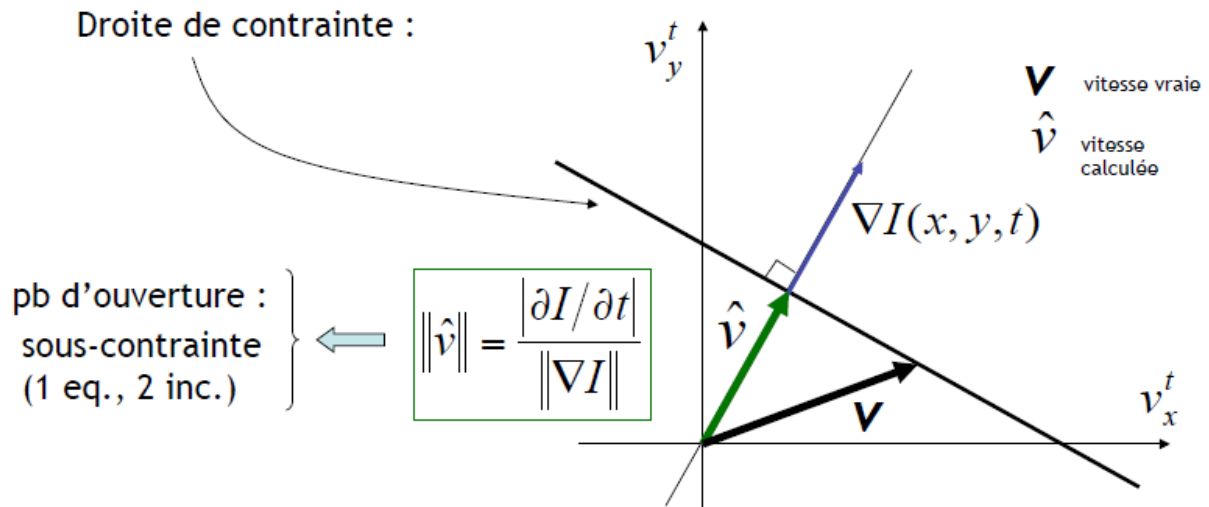
Equation de contrainte
du mouvement apparent (ECMA)
ou : équation du flot optique

avec

$$\left\{ \begin{array}{ll} \nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) & \text{gradient spatial} \\ v = (v_x^t, v_y^t) & \text{inconnues} \\ \frac{\partial I}{\partial t} & \text{gradient temporel} \end{array} \right.$$

Interprétation graphique ECMA

$$\nabla I \cdot \mathbf{v} + \frac{\partial I}{\partial t} = 0$$



1.3. Estimation de mouvement

Une des hypothèses qui doit être faite pour estimer le mouvement à partir du champ de mouvement apparent est que l'intensité image reste constante au cours du mouvement ou qu'elle varie d'une manière prédictible d'une image à l'autre [HOR 81], [VER 89].

L'hypothèse de conservation de l'intensité lumineuse en chaque point le long de la trajectoire du mouvement peut s'exprimer par l'équation DFD des différences entre les images déplacées (en anglais DFD=Displaced Frame Difference), c'est-à-dire entre les images aux instants t et $t+\Delta t$, avec $\Delta t=\pm 1$:

$$DFD = I(x+d_x, y+d_y, t+\Delta t) - I(x, y, t) = 0 \quad (1.3.1)$$

où $I(x, y, t)$ est l'intensité du point image $p=(x, y)$ à l'instant t et $d(p)=(d_x(p), d_y(p))$ est le vecteur déplacement correspondant, entre les instants t et $t+\Delta t$. On observe que:

- si les composantes de d ne sont pas entières, le calcul de la DFD en chaque pixel, nécessite une étape d'interpolation;
- si les composantes de d correspondent à la valeur réelle du déplacement, alors l'erreur d'estimation, donc la DFD, est nulle en chaque pixel.

L'estimation du mouvement réel à partir du mouvement apparent peut être abordée de deux manières différentes:

- l'estimation des vecteurs déplacement dans le plan image:

$$d(x, y, t) = (d_x(x, y, t), d_y(x, y, t)) \quad (1.3.2)$$

estimés entre les images à t et $t+1$;

- l'estimation des vecteurs vitesse:

$$v(x, y, t) = (v_x(x, y, t), v_y(x, y, t)) \quad (1.3.3)$$

Les vecteurs déplacement (respectivement vitesse) estimés peuvent varier en espace et en temps.

a) Estimation des vecteurs déplacement: L'estimation peut être vue comme un problème d'estimation de mouvement direct ou inverse ("avant" ou "arrière"), selon que l'estimation est réalisée entre les instants t et $t+1$ ou entre les instants t et $t-1$ (figure 1.3.1).

Dans le cas de l'estimation directe (avant) du mouvement, le problème se pose de la manière suivante: si on connaît les échantillons spatio-temporels $I_t(x, y)$ et $I_{t+1}(x, y)$, qui sont liés par la relation (hypothèse de conservation de l'intensité):

$$I_t(x, y) = I_{t+1}(x+d_x(x, y, t), y+d_y(x, y, t)), \quad (1.3.4)$$

on doit trouver le vecteur de déplacement:

$$d_t(x, y) = (d_{xd}(x, y, t), d_{yd}(x, y, t)) \quad (1.3.5)$$

Dans le cas de l'estimation inverse (arrière) du mouvement où les vecteurs de déplacement sont définis entre les instants t et $t-1$, on a la relation:

$$I_t(x, y) = I_{t-1}(x-d_{xi}(x, y, t), y-d_{yi}(x, y, t)), \quad (1.3.6)$$

En estimation de mouvement on utilise généralement l'estimation inverse (arrière). L'estimation avec compensation directe (avant) du mouvement est classiquement utilisée dans la compression prédictive de séquences d'images. Les valeurs prises par les déplacements d_x, d_y sont souvent réelles, ce qui nécessite une étape d'interpolation pour l'estimation du mouvement.

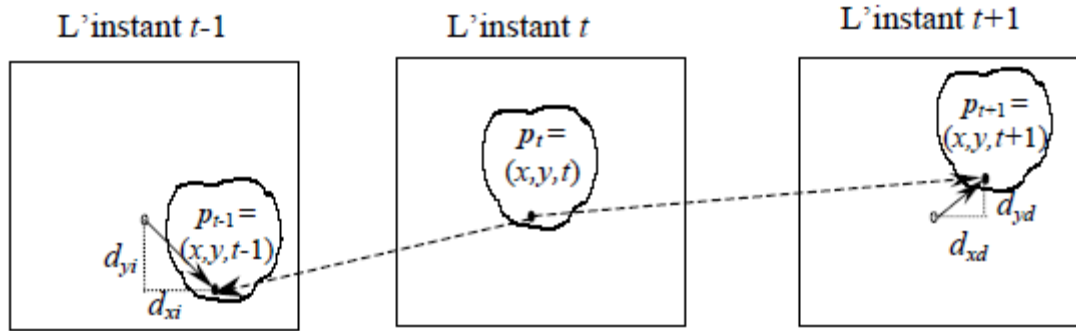


Figure 1.3.1. Estimation directe et inverse des vecteurs déplacement.

Dans le cas de l'estimation directe on a noté: $d_d(x, y, t) = (d_{xd}(x, y, t), d_{yd}(x, y, t))$.

Dans le cas de l'estimation inverse on a noté: $d_i(x, y, t) = (d_{xi}(x, y, t), d_{yi}(x, y, t))$.

b) Estimation des vecteurs vitesse: étant donnés les échantillons $I_t(x, y)$, on doit déterminer les vecteurs vitesse $v(x, y, t)$. On observe que si la vitesse reste constante dans l'intervalle Δt entre deux images et si Δt est petit, alors la vitesse estimée peut être assimilée au déplacement:

$$v_t(x, y) = \frac{d_t(x, y)}{\Delta t} \quad (1.3.7)$$

Pour un mouvement accéléré, on doit prendre en compte plus de deux images afin d'estimer correctement le champ de vitesse.

En conclusion, le champ de mouvement estimé peut être caractérisé par le champ de vecteurs vitesse ou par le champ de vecteurs déplacement (ou de correspondance). Le champ de vitesse estimé et le champ de déplacement sont identiques quand l'échantillonnage temporel de la séquence est constant. L'estimation du mouvement par la perspective de l'estimation du champ de vecteurs vitesse ou par l'estimation de vecteurs déplacement (ou de correspondance) représente donc deux approches équivalentes, quand on connaît le pas d'échantillonnage temporel de la séquence. Dans la suite on s'intéressera à l'estimation du champ de vecteurs déplacement.

L'estimation de mouvement est très sensible au bruit présent dans les images qui peut être interprété comme étant le résultat d'un mouvement dans la scène réelle.

L'estimation du mouvement est un problème mal-posé qui nécessite l'ajout de contraintes.

1.5. Méthodes de mise en correspondance

Les méthodes de mise en correspondance sont parmi les plus utilisées en pratique grâce à la simplicité de l'implantation physique du modèle de translation utilisé le plus souvent dans ces méthodes. Elles sont utilisées dans les standards de compression vidéo, comme H.261, MPEG-1, MPEG-2 ou MPEG-4.

Parmi les méthodes de mise en correspondance, on distingue deux classes:

- les méthodes de mise en correspondance dans le plan transformé;
- les méthodes de mise en correspondance dans le plan image.

1.5.1. Méthodes de mise en correspondance dans le plan transformé

Dans les méthodes de mise en correspondance dans le plan transformé, le champ de déplacement d est estimé par la mise en correspondance de blocs définis dans un plan transformé. La méthode la plus répandue est la méthode de corrélation de phase obtenue par la transformée de Fourier. Dans cette méthode, on calcule la corrélation de phase entre les transformées de Fourier de deux blocs correspondants dans les deux images successives [DUF 95]. On va noter: $\mathcal{F}\{I(x,y,t)\}=F_t(f_x,f_y)=F_t(f)$ la transformée de Fourier de l'image $I(x,y,t)$ à l'instant t , où $f=(f_x,f_y)$ représente les fréquences spatiales. La transformée de Fourier de l'image à l'instant $t+\Delta t$: $J(x,y,t+\Delta t)=I(x+d_x,y+d_y,t)$ est:

$$\mathcal{F}_{t+\Delta t}\{J(x,y,t+\Delta t)\}=F_{t+\Delta t}(f)=F_t(f)\cdot\exp\{-j\cdot 2\pi\cdot(f_x\cdot d_x+f_y\cdot d_y)\} \quad (1.5.1)$$

où $I(p,t)$ est l'intensité d'image au point $p=(x,y)$, à l'instant t , x et y représentent les coordonnées spatiales. $\Delta t=\pm 1$ représente l'intervalle de temps entre les deux images et $d(p)=(d_x(x),d_y(p))$ est le vecteur déplacement correspondant au point $p=(x,y)$.

Dans le cas d'un modèle de translation du mouvement, équivalent à l'équation (1.3.8) ou (1.3.9), la différence de phase entre les transformées de Fourier des deux blocs est donnée par:

$$\arg\{F_t(f) / F_{t+\Delta t}(f)\}=\Delta\varphi(f_x,f_y)=-2\cdot\pi\cdot(f_x\cdot d_x+f_y\cdot d_y) \quad (1.5.2)$$

L'équation (1.5.2) détermine un plan dans l'espace des variables f_x et f_y . Le mouvement entre les deux images consécutives peut être exprimé par la différence de phase entre deux paires de fréquences spatiales. Il est donné par un système linéaire des deux équations, correspondant ainsi à l'orientation du plan (équation (1.5.2)).

Les méthodes de corrélation de phase estiment le déplacement relatif entre deux images consécutives en utilisant une fonction normalisée d'inter-corrélation calculée dans l'espace de Fourier. Le résultat est un Dirac dont la position correspond au vecteur déplacement.

Un exemple d'une telle fonction ou coefficient d'inter-corrélation entre les images t et $t+\Delta t$ est:

$$c_{t,t+\Delta t}(x,y)=I_{t+\Delta t}(x+d_x,y+d_y)*I_t(-x,-y) \quad (1.5.3)$$

où $*$ représente l'opération de convolution. La transformée de Fourier de l'équation (1.5.3) fournit l'expression complexe du spectre inter-images, F^* étant la conjuguée de F :

$$C_{t,t+\Delta t}(f_x,f_y)=F_{t+\Delta t}(f_x,f_y)\cdot F_t^*(f_x,f_y) \quad (1.5.4)$$

En normalisant l'amplitude du spectre inter-images (1.5.4), on obtient la phase du spectre:

$$\arg\{C_{t,t+\Delta t}(f_x,f_y)\}=\frac{F_{t+\Delta t}(f_x,f_y)\cdot F_t^*(f_x,f_y)}{|F_{t+\Delta t}(f_x,f_y)\cdot F_t^*(f_x,f_y)|} \quad (1.5.5)$$

En supposant le modèle de translation (1.3.3) et en remplaçant (1.5.2) en (1.5.5), on obtient:

$$\arg\{C_{t,t+\Delta t}(f_x, f_y)\} = \exp\{-j \cdot 2\pi \cdot (f_x \cdot d_x + f_y \cdot d_y)\} \quad (1.5.6)$$

La transformée de Fourier inverse (1.5.6), donne la fonction de corrélation de phase:

$$\arg\{c_{t,t+\Delta t}(x, y)\} = \delta(x - d_x, y - d_y) \quad (1.5.7)$$

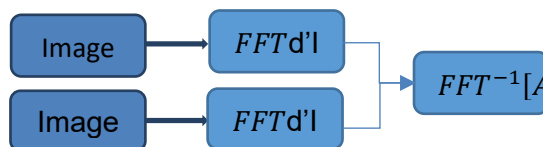
où δ est l'impulsion delta Dirac: $\delta(q) = \begin{cases} 1, & \text{pour } q = 0 \\ 0, & \text{en rest} \end{cases}$

En pratique, la transformée de Fourier est remplacée par la Transformée de Fourier (TF) discrète, pour laquelle il y a des algorithmes rapides de calcul. Un exemple d'un tel algorithme utilisant la corrélation de phase est décrit dans (1.5.8).

1. Calcul de TF discrète 2D des blocs correspondants dans les images t et $t + \Delta t$;
 2. Calcul de la phase du spectre inter-images (1.5.6);
 3. Calcul de la TF inverse de la relation (1.5.6), afin d'obtenir la fonction de corrélation de phase (1.5.7).
 4. Détection de la position du pic du maximum de corrélation de phase, donnant la valeur estimée du déplacement $\hat{d} = (\hat{d}_x, \hat{d}_y)$.
- (1.5.8)

Dans le cas idéal, la fonction de corrélation de phase doit présenter un seul maximum (pic) qui indique le déplacement relatif entre les deux blocs. En pratique, la

Algorithme de Cross-Correlation (mise en correspondance dans le domaine de transformée)



Déplacement
de a par rapport à b

(estimer le vecteur déplacement (d_x, d_y))

Fig. 2.1 l'algorithme de recalage des images
par corrélation croisée.

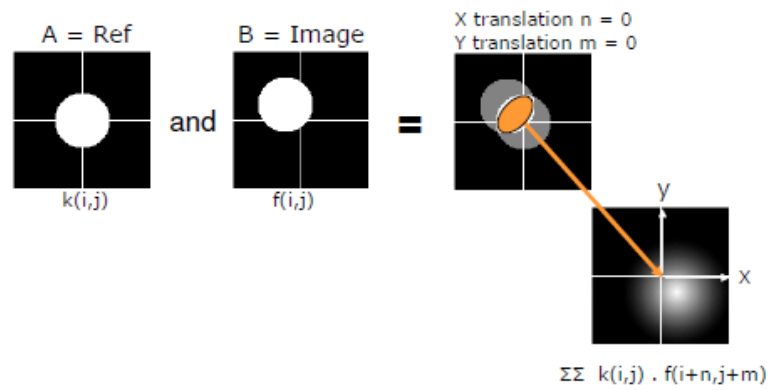


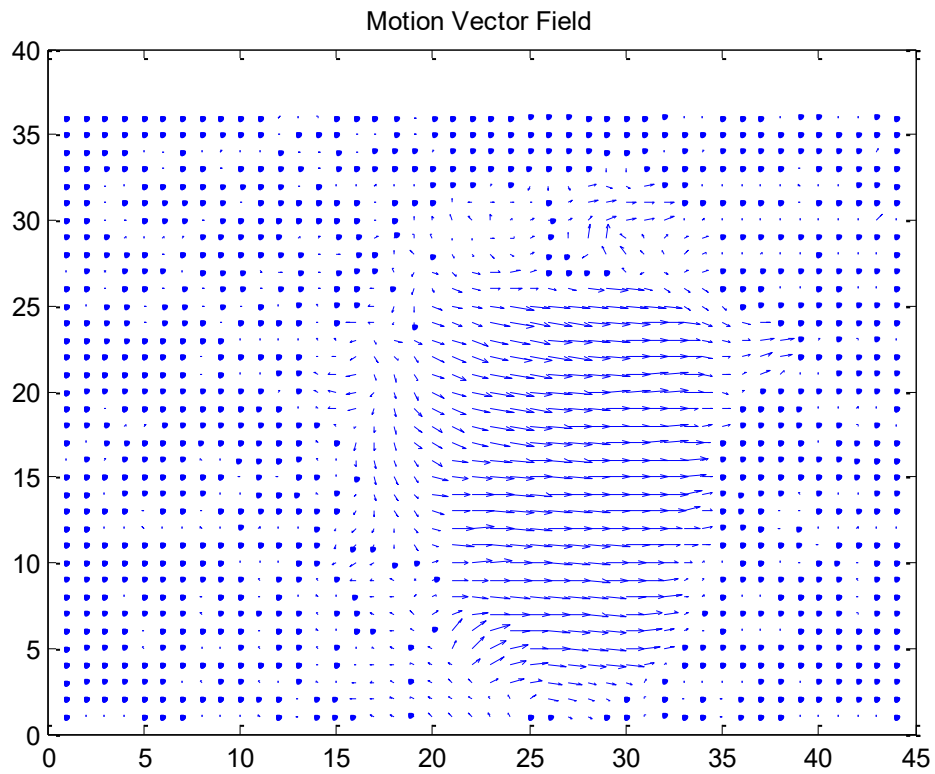
Fig. 2.2 Principe d'alignement par corrélation croisée.

Exemples sous Matlab:**Estimation par flux optique**

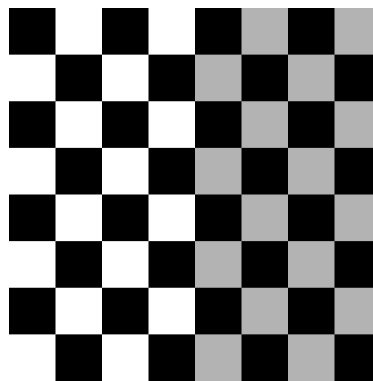
fichier:

D:\Enseignement\Modules\vision_artificielle\motion_estimation\SubME_1.6

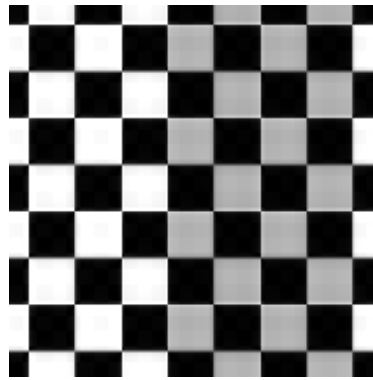




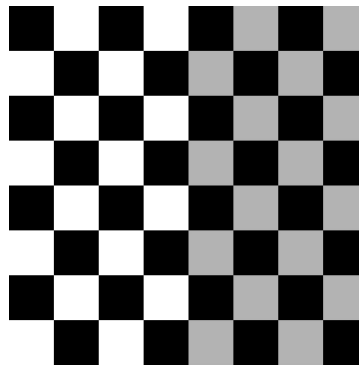
Exemple 2: mise en correspondance des points (cross-correlation method)



(a) Image originale: Image checkerboard de taille 400×400 pixels.



(b) Image Checkerboard déplacée ($dx = 33.4857$, $dy = 13.7383$).



(c) Image Checkerboard alignée par CC (Cross-Correlation)

Exemple 3: Efficient subpixel image registration by cross-correlation (alignement par corrélation croisée)

```
f = im2double(imread('cameraman.tif'));

deltar = -3.48574;
deltac = 8.73837;

phase = 2;

[nr,nc]=size(f);

Nr = ifftshift((-fix(nr/2):ceil(nr/2)-1));
Nc = ifftshift((-fix(nc/2):ceil(nc/2)-1));

[Nc,Nr] = meshgrid(Nc,Nr);

g = ifft2(fft2(f).*exp(1i*2*pi*(deltar*Nr/nr+deltac*Nc/nc))).*exp(-1i*phase);

figure(1);

subplot(1,2,1);

imshow(abs(f));

title('Reference image, f(x,y)')
```

```
subplot(1,2,2);
imshow(abs(g));
title('Shifted image, g(x,y)')
```

Reference image, $f(x,y)$ Shifted image, $g(x,y)$ 

```
usfac = 100;
[output, Greg] = dftregistration(fft2(f),fft2(g),usfac);
display(output),
```

```
-----
output =
```

```
0.0007  2.0000 -3.4900  8.7400
```

```
figure(1);
subplot(1,2,1);
imshow(abs(f));
title('Reference image, f(x,y)')
subplot(1,2,2);
imshow(abs(iff2(Greg)));
title('Registered image, gr(x,y)')
```


Reference image, $f(x,y)$ Registered image, $gr(x,y)$ 

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% la fonction dftregistration

function [output Greg] = dftregistration(buf1ft,buf2ft,usfac)

```
% function [output Greg] = dftregistration(buf1ft,buf2ft,usfac);
% Efficient subpixel image registration by crosscorrelation. This code
% gives the same precision as the FFT upsampled cross correlation in a
% small fraction of the computation time and with reduced memory
% requirements. It obtains an initial estimate of the crosscorrelation peak
% by an FFT and then refines the shift estimation by upsampling the DFT
% only in a small neighborhood of that estimate by means of a
% matrix-multiply DFT. With this procedure all the image points are used to
% compute the upsampled crosscorrelation.
% Manuel Guizar - Dec 13, 2007
```

```
% Portions of this code were taken from code written by Ann M. Kowalczyk
% and James R. Fienup.
% J.R. Fienup and A.M. Kowalczyk, "Phase retrieval for a complex-valued
% object by using a low-resolution image," J. Opt. Soc. Am. A 7, 450-458
% (1990).
```

```
% Citation for this algorithm:
% Manuel Guizar-Sicairos, Samuel T. Thurman, and James R. Fienup,
% "Efficient subpixel image registration algorithms," Opt. Lett. 33,
% 156-158 (2008).
```

```
% Inputs
% buf1ft    Fourier transform of reference image,
%           DC in (1,1) [DO NOT FFTSHIFT]
% buf2ft    Fourier transform of image to register,
%           DC in (1,1) [DO NOT FFTSHIFT]
% usfac     Upsampling factor (integer). Images will be registered to
%           within 1/usfac of a pixel. For example usfac = 20 means the
%           images will be registered within 1/20 of a pixel. (default = 1)
```

```
% Outputs
```

```

% output = [error,diffphase,net_row_shift,net_col_shift]
% error      Translation invariant normalized RMS error between f and g
% diffphase   Global phase difference between the two images (should be
%             zero if images are non-negative).
% net_row_shift net_col_shift   Pixel shifts between images
% Greg        (Optional) Fourier transform of registered version of buf2ft,
%             the global phase difference is compensated for.

% Default usfac to 1
if exist('usfac')~=1, usfac=1; end

% Compute error for no pixel shift
if usfac == 0,
    CCmax = sum(sum(buf1ft.*conj(buf2ft)));
    rfzero = sum(abs(buf1ft(:)).^2);
    rgzero = sum(abs(buf2ft(:)).^2);
    error = 1.0 - CCmax.*conj(CCmax)/(rgzero*rfzero);
    error = sqrt(abs(error));
    diffphase=atan2(imag(CCmax),real(CCmax));
    output=[error,diffphase];

% Whole-pixel shift - Compute crosscorrelation by an IFFT and locate the
% peak
elseif usfac == 1,
    [m,n]=size(buf1ft);
    CC = ifft2(buf1ft.*conj(buf2ft));
    [max1,loc1] = max(CC);
    [max2,loc2] = max(max1);
    rloc=loc1(loc2);
    cloc=loc2;
    CCmax=CC(rloc,cloc);
    rfzero = sum(abs(buf1ft(:)).^2)/(m*n);
    rgzero = sum(abs(buf2ft(:)).^2)/(m*n);
    error = 1.0 - CCmax.*conj(CCmax)/(rgzero(1,1)*rfzero(1,1));
    error = sqrt(abs(error));
    diffphase=atan2(imag(CCmax),real(CCmax));
    md2 = fix(m/2);
    nd2 = fix(n/2);
    if rloc > md2
        row_shift = rloc - m - 1;
    else
        row_shift = rloc - 1;
    end

    if cloc > nd2
        col_shift = cloc - n - 1;
    else
        col_shift = cloc - 1;
    end
    output=[error,diffphase,row_shift,col_shift];

% Partial-pixel shift
else

    % First upsample by a factor of 2 to obtain initial estimate
    % Embed Fourier data in a 2x larger array
    [m,n]=size(buf1ft);
    mlarge=m*2;
    nlarge=n*2;
    CC=zeros(mlarge,nlarge);
    CC(m+1-fix(m/2):m+1+fix((m-1)/2),n+1-fix(n/2):n+1+fix((n-1)/2)) = ...
        fftshift(buf1ft).*conj(fftshift(buf2ft));

    % Compute crosscorrelation and locate the peak
    CC = ifft2(fftshift(CC)); % Calculate cross-correlation
    [max1,loc1] = max(CC);
    [max2,loc2] = max(max1);
    rloc=loc1(loc2);cloc=loc2;

```

```

CCmax=CC(rloc,cloc);

% Obtain shift in original pixel grid from the position of the
% crosscorrelation peak
[m,n] = size(CC); md2 = fix(m/2); nd2 = fix(n/2);
if rloc > md2
    row_shift = rloc - m - 1;
else
    row_shift = rloc - 1;
end
if cloc > nd2
    col_shift = cloc - n - 1;
else
    col_shift = cloc - 1;
end
row_shift=row_shift/2;
col_shift=col_shift/2;

% If upsampling > 2, then refine estimate with matrix multiply DFT
if usfac > 2,
    %%% DFT computation %%%
    % Initial shift estimate in upsampled grid
    row_shift = round(row_shift*usfac)/usfac;
    col_shift = round(col_shift*usfac)/usfac;
    dftshift = fix(ceil(usfac*1.5)/2); %% Center of output array at dftshift+1
    % Matrix multiply DFT around the current shift estimate
    CC =
conj(dftups(buf2ft.*conj(buf1ft),ceil(usfac*1.5),ceil(usfac*1.5),usfac,...
    dftshift-row_shift*usfac,dftshift-col_shift*usfac))/(md2*nd2*usfac^2);
    % Locate maximum and map back to original pixel grid
    [max1,loc1] = max(CC);
    [max2,loc2] = max(max1);
    rloc = loc1(loc2); cloc = loc2;
    CCmax = CC(rloc,cloc);
    rg00 = dftups(buf1ft.*conj(buf1ft),1,1,usfac)/(md2*nd2*usfac^2);
    rf00 = dftups(buf2ft.*conj(buf2ft),1,1,usfac)/(md2*nd2*usfac^2);
    rloc = rloc - dftshift - 1;
    cloc = cloc - dftshift - 1;
    row_shift = row_shift + rloc/usfac;
    col_shift = col_shift + cloc/usfac;

% If upsampling = 2, no additional pixel shift refinement
else
    rg00 = sum(sum( buf1ft.*conj(buf1ft) ))/m/n;
    rf00 = sum(sum( buf2ft.*conj(buf2ft) ))/m/n;
end
error = 1.0 - CCmax.*conj(CCmax)/(rg00*rf00);
error = sqrt(abs(error));
diffphase=atan2(imag(CCmax),real(CCmax));
% If its only one row or column the shift along that dimension has no
% effect. We set to zero.
if md2 == 1,
    row_shift = 0;
end
if nd2 == 1,
    col_shift = 0;
end
output=[error,diffphase,row_shift,col_shift];
end

% Compute registered version of buf2ft
if (nargout > 1)&&(usfac > 0),
    [nr,nc]=size(buf2ft);
    Nr = ifftshift([-fix(nr/2):ceil(nr/2)-1]);
    Nc = ifftshift([-fix(nc/2):ceil(nc/2)-1]);
    [Nc,Nr] = meshgrid(Nc,Nr);
    Greg = buf2ft.*exp(i*2*pi*(-row_shift*Nr/nr-col_shift*Nc/nc));
    Greg = Greg*exp(i*diffphase);
end

```

```

elseif (nargout > 1)&&(usfac == 0)
    Greg = buf2ft*exp(i*diffphase);
end
return

function out=dftups(in,nor,noc,usfac,roff,coff)
% function out=dftups(in,nor,noc,usfac,roff,coff);
% Upsampled DFT by matrix multiplies, can compute an upsampled DFT in just
% a small region.
% usfac      Upsampling factor (default usfac = 1)
% [nor,noc]   Number of pixels in the output upsampled DFT, in
%             units of upsampled pixels (default = size(in))
% roff, coff  Row and column offsets, allow to shift the output array to
%             a region of interest on the DFT (default = 0)
% Recieves DC in upper left corner, image center must be in (1,1)
% Manuel Guizar - Dec 13, 2007
% Modified from dftus, by J.R. Fienup 7/31/06

% This code is intended to provide the same result as if the following
% operations were performed
%   - Embed the array "in" in an array that is usfac times larger in each
%     dimension. ifftshift to bring the center of the image to (1,1).
%   - Take the FFT of the larger array
%   - Extract an [nor, noc] region of the result. Starting with the
%     [roff+1 coff+1] element.

% It achieves this result by computing the DFT in the output array without
% the need to zeropad. Much faster and memory efficient than the
% zero-padded FFT approach if [nor noc] are much smaller than [nr*usfac nc*usfac]

[nr,nc]=size(in);
% Set defaults
if exist('roff')~=1, roff=0; end
if exist('coff')~=1, coff=0; end
if exist('usfac')~=1, usfac=1; end
if exist('noc')~=1, noc=nc; end
if exist('nor')~=1, nor=nr; end
% Compute kernels and obtain DFT by matrix products
kernc=exp((-i*2*pi/(nc*usfac))*( ifftshift([0:nc-1]).' - floor(nc/2) )*( [0:noc-1]
- coff ));
kernr=exp((-i*2*pi/(nr*usfac))*( [0:nor-1].' - roff )*( ifftshift([0:nr-1]) -
floor(nr/2) ));
out=kernr*in*kernc;
return

```

reference image



target image



Registrated image



Exemple 4: Find Image Rotation and Scale Using Automated Feature Matching

- Step 1: Read Image
- Step 2: Resize and Rotate the Image
- Step 3: Find Matching Features Between Images
- Step 4: Estimate Transformation
- Step 5: Solve for Scale and Angle
- Step 6: Recover the Original Image

%Step1

original = imread('cameraman.tif');

imshow(original);

text(size(original,2),size(original,1)+15, ...

'Image courtesy of Massachusetts Institute of Technology', ...

'FontSize',7,'HorizontalAlignment','right');



Image courtesy of Massachusetts Institute of Technology

%Step 2: Resize and Rotate the Image

```
scale = 0.7;
```

```
J = imresize(original, scale); % Try varying the scale factor.
```

```
theta = 30;
```

```
distorted = imrotate(J,theta); % Try varying the angle, theta.
```

```
figure, imshow(distorted)
```



%Step 3: Find Matching Features Between Images

```
%Detect features in both images.
```



```

ptsOriginal = detectSURFFeatures(original);
ptsDistorted = detectSURFFeatures(distorted);

%Extract feature descriptors.

[featuresOriginal, validPtsOriginal] = extractFeatures(original, ptsOriginal);
[featuresDistorted, validPtsDistorted] = extractFeatures(distorted, ptsDistorted);

%Match features by using their descriptors.

indexPairs = matchFeatures(featuresOriginal, featuresDistorted);

%Retrieve locations of corresponding points for each image.

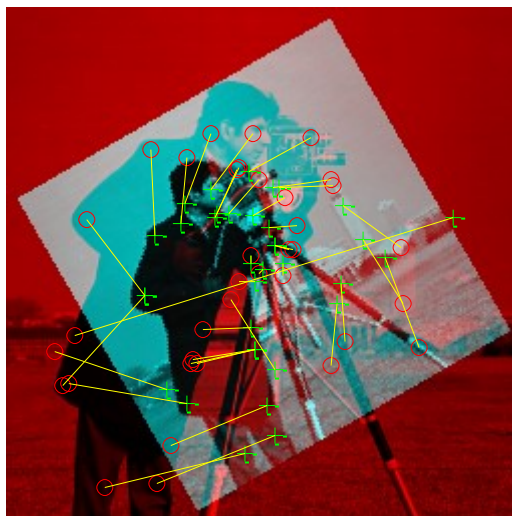
matchedOriginal = validPtsOriginal(indexPairs(:,1));
matchedDistorted = validPtsDistorted(indexPairs(:,2));

%Show point matches. Notice the presence of outliers.

figure;
showMatchedFeatures(original,distorted,matchedOriginal,matchedDistorted);
title('Putatively matched points (including outliers)');

```

Putatively matched points (including outliers)

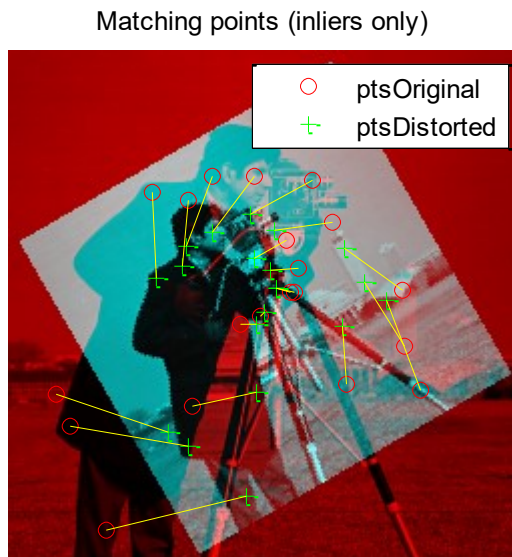


%Step 4: Estimate Transformation

```
[tform, inlierDistorted, inlierOriginal] = estimateGeometricTransform(...
    matchedDistorted, matchedOriginal, 'similarity');
```

%Display matching point pairs used in the computation of the transformation matrix.

```
figure;
showMatchedFeatures(original,distorted, inlierOriginal, inlierDistorted);
title('Matching points (inliers only)');
legend('ptsOriginal','ptsDistorted');
```

**%Step 5: Solve for Scale and Angle**

%Use the geometric transform, TFORM, to recover the scale and angle. Since we computed the %transformation from the distorted to the original image, we need to compute its inverse to recover %the distortion.

```
%Let
tx=0.7;
ty=0.7;
sc = scale*cos(theta)
ss = scale*sin(theta)
```

```
%Then
Tinv = [sc -ss 0;
        ss  sc 0;
        tx ty 1]

%where tx and ty are x and y translations, respectively.
%Compute the inverse transformation matrix.
```

```
Tinv = tform.invert.T;
```

```
ss = Tinv(2,1);
```

```
sc = Tinv(1,1);
```

```
scale_recovered = sqrt(ss*ss + sc*sc)
```

```
theta_recovered = atan2(ss,sc)*180/pi
```

RÉSULTATS:

```
sc =
```

```
0.1080
```

```
ss =
```

```
-0.6916
```

```
Tinv =
```

```
0.1080  0.6916    0
```

```
-0.6916  0.1080    0
```

```
0.7000  0.5000  1.0000
```

```
scale_recovered =
```

```
0.6994
```

```
theta_recovered =
```

```
30.0321
```

%Step 6: Recover the Original Image

%Recover the original image by transforming the distorted image.

```
outputView = imref2d(size(original));
```

```
recovered = imwarp(distorted,tform,'OutputView',outputView);
```

%Compare recovered to original by looking at them side-by-side in a montage.

```
figure, imshowpair(original,recovered,'montage')
```



Exemple 5: registering-an-image-using-normalized-cross-correlation

<https://www.mathworks.com/examples/image/mw/images-ex81619570-registering-an-image-using-normalized-cross-correlation>