TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

Projet de Réalité Augmentée
Partie I - Détection de coin

1. Introduction

La transformation d'une image quelle qu'elle soit a pour vocation de modifier à loisir les propriétés des pixels, que cela touche à leur valeur et donc à l'intensité (ou la couleur) d'une région de l'image ou bien à leur position dans la « matrice image » correspondant à la forme qui sera perçue in fine. Ce travail sur les pixels peut être à fin corrective ou méliorative afin de compenser des imperfections apparues lors de la prise d'image (par exemple des distorsions géométriques) de sorte que l'on aura une image optimisée au bout du compte, ou bien à fin de pure modification ou d'assemblage, notamment pour créer des effets spéciaux, artistiques, de profondeur, mettre en correspondance des images d'un même objet mais différentes en leur instant de prise ou en leur point de vue ou encore remplacer un contenu (montage).

Dans le cadre de ce projet nous nous intéresserons à cette dernière application et nous efforcerons de remplacer le contenu d'une feuille A4 qu'une personne déplace au cours d'une vidéo. Cette dernière apparaît comme une succession d'images de point de vue fixe (plongeant/oblique) mais dont les différentes dates de prises de vue correspondent à différentes positions de la feuille. L'enjeu est donc tout d'abord de pouvoir détecter les coins de la feuille A4 sur une image de la vidéo : il s'agit de la première partie de ce projet.

2. Démarche algorithmique

Nous allons donc sur chaque nouvelle image de la vidéo détecter le plus précisément possible les quatre coins de la feuille qui se déplace. Sur chaque nouvelle image les coins vont donc bouger et il faudra appliquer à nouveau notre algorithme de détection.

L'idée générale de la détection de coin et plus généralement de la détection de contour est d'étudier le gradient de l'image considérée. En effet un contour ou un coin constitue une variation brutale au niveau des pixels formant le motif en question. Cela correspond à une valeur élevée de gradient à l'endroit considéré. Ainsi, en calculant la matrice du gradient de l'image, on va pouvoir repérer les valeurs d'intensité remarquable et appliquer un seuil. Ce seuil correspond à une valeur minimale de gradient d'intensité qu'un pixel doit dépasser pour être considéré comme un potentiel coin.

Pour chaque nouvelle image de la vidéo, nous allons donc calculer son gradient et déterminer un seuil de valeur. Il faut calculer à la fois le gradient horizontal et le gradient vertical pour obtenir le gradient complet d'une image, car elle possède deux direction (deux coordonnées, objet à deux dimensions). Pour un contour "simple" il aurait pu suffire de calculer le gradient dans une seule direction (la plus adaptée au contour en particulier) mais un coin constitue un pic de gradient d'intensité dans les deux directions de l'images.

Toutefois, simplement regarder le gradient d'une image pour y détecter des coins n'aboutit pas forcément et conduit quasi nécessairement à des erreurs. Il existe des méthodes de détections plus élaborées, spécifiquement dédiées à la détection de point d'intérêt comme les coins. Pour ce projet, la méthode choisie est le détecteur de Harris.

Cette méthode (une amélioration du détecteur de Moravec) est basée sur un calcul de la matrice de covariance du gradient de l'image, pour ainsi quantifier le comportement local (au voisinage du pixel considéré) du gradient.

Pour remplir la matrice de covariance que l'on notera M, on va premièrement calculer les gradients de l'image I selon les deux directions x et y (que l'on notera I_x et I_y). On pourrait simplement convoluer I avec un filtre dérivateur comme le filtre 0,5*[1 0 -1] mais cela rend notre gradient trop sensible au bruit et cela fige sa taille. Pour contrer ce problème, on va simuler la dérivation en multipliant directement I par la dérivée selon x et y d'une gaussienne G. On va donc appliquer un filtre gaussien à I, qui va donc lisser le signal et atténuer les hautes fréquences (filtre passe-bas). Cette gaussienne est caractérisé par son écart type σ et s'écrit donc :

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Il est important d'échantillonner x et y sur [-3σ ;3σ] pour que le filtre qu'on applique profite de toutes les propriétés de la gaussienne. De même, il faut correctement choisir σ : s'il est trop faible on va se rapprocher d'un pic de Dirac et avoir un filtrage peu impactant mais si on le prend trop élevé le filtre s'élargit trop et atténue moins bien le signal. On va donc ensuite dériver G selon x et y :

$$\begin{cases} \frac{\partial G(x,y)}{\partial x} = \frac{-x}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \\ \frac{\partial G(x,y)}{\partial y} = \frac{-y}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \end{cases}$$

Le calcul de I_x et I_y peut donc se faire :

$$\begin{cases} I_x = I * \frac{\partial G(x,y)}{\partial x} \\ I_y = I * \frac{\partial G(x,y)}{\partial y} \end{cases}$$

(Ici « * » désigne le produit de convolution)

La matrice de covariance de chaque point (x,y) va alors pouvoir être remplie avec les expressions suivantes :

$$M(x,y) = \begin{bmatrix} A & B \\ B & C \end{bmatrix} \qquad \text{avec} \begin{cases} A = \int_\Omega I_x^2 \\ B = \int_\Omega I_x I_y \\ C = \int_\Omega I_y^2 \end{cases}$$

Ainsi pour chaque pixel (x,y) on définit une matrice de covariance pour un voisinage Ω autour de ce pixel.

Nous allons maintenant aborder la dernière composante du détecteur de Harris, qui nous donne des valeurs directement exploitables sous forme d'une matrice nous indiquant quels pixels sont des coins. On va alors faire mieux que simplement retenir les valeurs élevées de gradient : on va regarder quels pixels peuvent être des potentiels coins en regardant aussi ses voisins proches. Pour cela, on définit une matrice (c'est cela qu'on appelle communément le détecteur de Harris) notée D, où chaque coefficient D(x,y) nous apporte une information précise et quantifiable sur la nature du pixel en question (coin, contour, pixel « banal »). On a :

$$D(x,y) = \det\big(M(x,y)\big) - k\ tr(M(x,y))^2$$

où « det » désigne le déterminant de M, « tr » désigne la trace de M et k est un paramètre que l'on fixe après quelques essais à 0,05.

Après avoir calculé les D(x,y) pour chaque pixel de l'image, on obtient la matrice de Harris D. C'est par l'exploitation de cette matrice qu'on va pouvoir déterminer les coins de la feuille. En effet, pour un pixel ayant une valeur D(x,y), on va avoir trois cas possibles :

- D(x,y) < 0 : le gradient au voisinage du pixel est élevé mais que dans une ou peu de direction, c'est une zone contour.
- D(x,y) ≈ 0 : le gradient au voisinage du pixel est faible, on est sur une zone d'intensité peu variable.
- D(x,y) > 0 : le gradient au voisinage du pixel est élevé, c'est un coin.

Ainsi on obtient une matrice D facile à manipuler, qui nous indique directement la présence de coin. Il suffit de parcourir D et de récupérer les coordonnées des pixels qui ont une valeur positive loin de 0 dans la matrice de Harris.

Nous avons donc vu le principe théorique de la détection des coins sur une image. Pour chaque image de la vidéo, on va appliquer ces étapes, calculer la matrice D et ainsi repérer les coordonnées des coins. Nous allons maintenant voir comment implémenter ces étapes théoriques sur Matlab pour obtenir un résultat clair, c'est-à-dire obtenir les coordonnées des quatre coins de la feuille qui se déplace sur la vidéo. Il est important d'obtenir un résultat simple pour pouvoir l'utiliser

3. Implémentation

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

En traitement d'image, on utilise le plus souvent des images en couleur. Cela revient donc à manipuler des matrices M*N où pour chaque pixel trois canaux de couleurs sont accessible. Ainsi on va pouvoir utiliser plusieurs espaces colorimétriques comme la palette RGB qui est classique (façon « naturelle » de représenter les couleurs) mais pas optimale, ou encore l'espace YCbCr que nous avons choisi d'utiliser. Tous les calculs évoqués précédemment sont des calculs matriciels d'algèbre, des calculs de convolution. On a donc besoin d'une seule valeur par pixel pour effectuer ces calculs correctement. C'est là que le choix de l'espace YCbCr prend sens : la composante Y correspond en une seule valeur à la somme des trois composantes RGB. Ainsi en ne sélectionnant que la composante Y, notre image (anciennement M*N*3) devient une « simple » matrice M*N que l'on sait manipuler sans souci.

Première étape du traitement de chaque image de la vidéo, calculer le gradient de l'image traitée, en calculant la dérivée de l'intensité horizontale et verticale. Comme on a pu le voir, on va simuler la dérivation en appliquant (convolution) un filtre gaussien lui-même dérivé selon x et selon y. On va donc créer une fonction **intensityGradient** ayant pour argument d'entrée l'image I (ou plutôt uniquement son canal Y comme on a pu le voir, dans la suite on considère que quand on parle de I on parle uniquement du canal Y de l'image) et sigma le paramètre de la gaussienne utilisée comme filtre (ici on le prend égal à 2). Cette fonction renvoie les gradients d'intensités Ix et Iy que l'on peut alors directement exploiter. On échantillonne bien les vecteurs x et y entre -3*sigma et +3*sigma pour conserver les propriétés du filtre gaussien, et il ne reste plus qu'à convoluer I par les dérivées des gaussiennes selon x (Gx) et selon y (Gy) pour obtenir Ix et Iy. On utilise la commande conv2 qui effectue une convolution en deux dimensions.

Passons maintenant à l'implémentation du détecteur de Harris en lui-même. On créer une fonction **harrisDetect** qui prend en argument les gradients selon x et y, le paramètre k que l'on va prendre égal à 0,05 et toujours sigma le paramètre de la gaussienne. En effet, on ne va pas réutiliser la même gaussienne (ou plutôt pas le même sigma) pour garder la possibilité de changer sigma uniquement pour le détecteur. En suivant la démarche théorique, on calcule dans cette fonction les coefficients de la matrice de covariance en convoluant les carrés de Ix et Iy, le produit Ix*Iy par G, la gaussienne définie au début de la fonction. Nous avons alors les trois différents coefficient de notre matrice de covariance, que l'on note simplement a,b,c. Attention, les variables a,b,c sont elles-mêmes des matrices car tout notre espace (x,y) est discrétisé.

Ainsi on peut calculer notre matrice de Harris que l'on notera D. Son expression en fonction des paramètres a,b,c,k et sigma est alors :

$$D=a.*c-b.^2-k*(a+c).^2 ;$$

La fonction **harrisDetect** nous renvoie donc directement une matrice D ayant les mêmes propriétés que la théorie nous indique.

Cependant d'un point de vue du projet, la détection des coins n'est pas terminée. En effet dans cette matrice D, beaucoup de cases ont la valeur numérique d'un coin potentiel.

Pour résoudre ce problème, nous allons utiliser les informations issues du traitement de l'image précédente. En effet, à la fin de chaque image, si notre traitement marche on aura récupéré un ensemble de quatre points (x,y) correspondant aux coordonnées des quatre coins de la feuille. L'hypothèse à faire pour pouvoir bénéficier des données de l'image précédente consiste à considérer que, en passant à l'image d'après, les coins de la feuille ne se déplace que d'une petite distance. Cette hypothèse est tout à fait cohérente : sur la vidéo, la feuille ne quitte pas la table, n'effectue pas de mouvement brusque et vu que la vidéo contient 325 images, les coins ne bougent que très peu entre deux images.

Cependant ce raisonnement par récurrence a besoin d'une initialisation. Il faut donc faire le compromis de faire intervenir l'utilisateur au début de la vidéo. Ainsi, en lançant l'algorithme, l'utilisateur va devoir cliquer sur les quatre coins de la première image de la vidéo pour transmettre les coordonnées des quatre premiers coins à Matlab. Cette acquisition s'effectue à l'aide de la commande getpts de Matlab.

Nous allons donc pouvoir nous servir de cette information dans nos algorithmes pour localiser dans la matrice de Harris quatre zones où les coins sont susceptibles d'être d'après l'image précédente. Ces zones sont carrées et centrées sur chaque coin. Nous créons donc une fonction **zoneCoin** qui va prendre en entrée un paramètre delta qui est en fait la moitié d'un côté du carré que l'on veut créer et un point P qui va en fait être un des quatre coins. Cette fonction va nous renvoyer quatre points sous la forme de quatre couple (x,y) qui correspondent aux quatre sommets du carré créé.

On a donc réduit le problème du parcours de la matrice de Harris au simple parcours de quatre sous-matrice de taille 50*50. C'est une taille ridicule comparée à la taille de l'image (1080x1920 pixels). Par conséquent, il est très peu probable que deux coins se retrouve dans une aussi petite zone. Nous pouvons alors considérer que le maximum de chaque sous-matrice de Harris est donc un coin. Ce procédé permet non seulement d'améliorer la fiabilité de l'algorithme mais aussi de le rendre plus rapide.

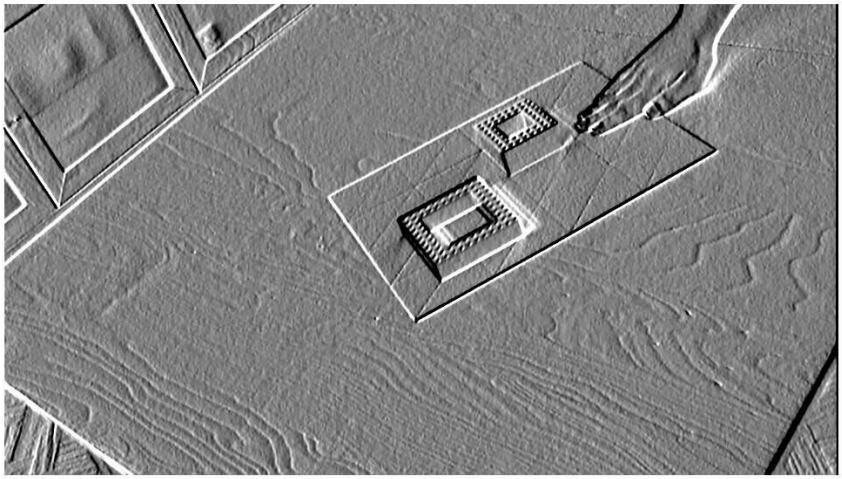
Il ne reste plus qu'à implémenter une fonction **maxCoin** qui prend en entrée les coordonnées des quatre sommets d'une sous-matrice et la matrice de Harris D, et qui nous renvoie les coordonnées du maximum de la sous-matrice, c'est-à-dire les coordonnées d'un coin. On applique cette fonction dans les quatre sous-matrices, et on peut donc stocker les coordonnées des quatre coins de la feuille pour une image donnée. Ces coordonnées vont être réutilisées à l'image d'après pour déterminer la position des futurs coins, et ainsi de suite.

4. Résultats

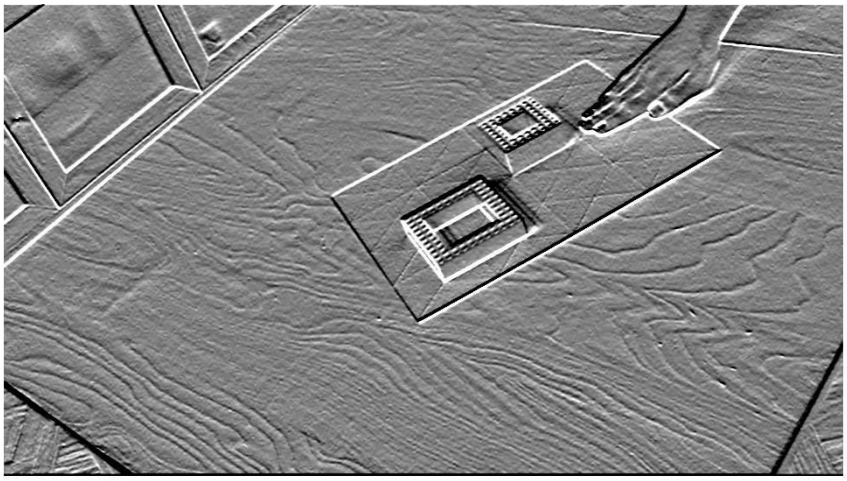
On donne le résultat du gradient d'intensité du premier frame de la vidéo :

Gradient d'intensité suivant x

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin



Gradient d'intensité selon y



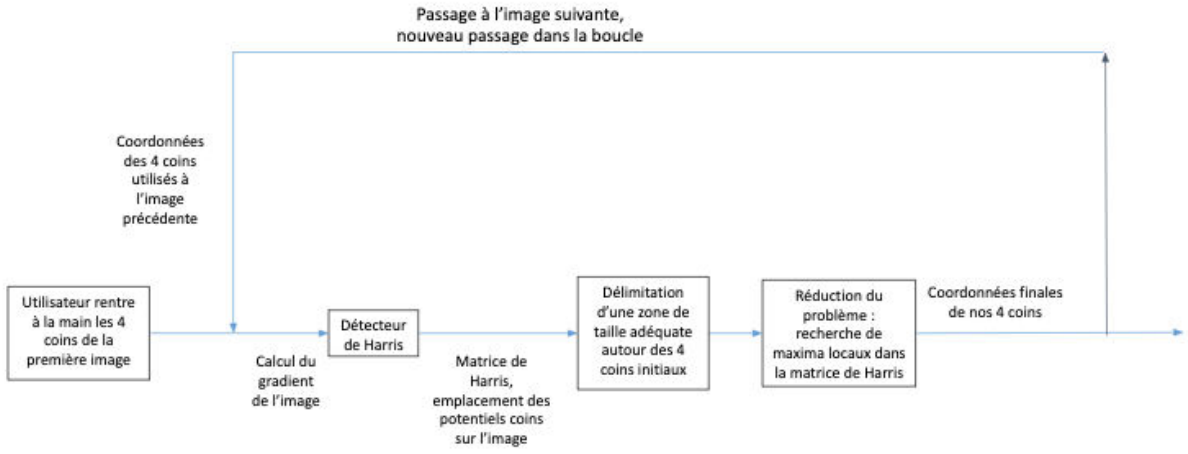
Ce qui nous donne le détecteur de Harris suivant :

Détecteur de Harris

Il est donc clair que la détection des coins de la feuille fonctionne. On remarquera que les coins des pièces de lego ne sont eux pas très bien détectées dû aux nombreuses irrégularités de surface. En effet, les paramètres choisis pour ce détecteur sont optimisés pour une bonne détection des coins de la feuille et non des legos. Enfin, notre détecteur n'est pas optimisé car il balaie la totalité de chaque frame dans le code final. Il serait préférable de procéder a du fenêtrage qui appliquerait alors le détecteur que dans une fenêtre de pixels réduit. Cependant, nous avons choisi de nous concentrer sur l'optimisation de l'homographie ainsi que le développement du 3D et n'avons donc pas eu le temps d'optimiser la détection.

5. Conclusion

Ainsi, nous avons créé un traitement qui s'applique sur chaque image de la vidéo et qui permet d'extraire les coordonnées des quatre coins de la feuille. Ce traitement utilise les informations issues de l'image précédente et nécessite donc une initialisation manuelle par l'utilisateur. En remplissant la matrice du détecteur de Harris, on peut avoir l'information de la localisation des potentiels coins d'une image, et on affine cette recherche en réduisant le problème en sous-matrice, ce qui accélère et fiabilise la recherche des quatre coins de la feuille. Nous pouvons résumer le principe de cet algorithme de détection de contours sous la forme du schéma-bloc suivant :



6. Annexe

```
function [Gradx,Grady] =
intensityGradient(I,sigma)

% Summary : On determine le gradient
d'intensit√© d'une image
%
% Description : A l'aide d'un produit
de convolution on determine le
% gradient d'intensite d'une image I.
La convolution se fait grace aux
% derivees partielles d'une gaussienne
Gx et Gy. On peut donc jouer sur le
% parametre sigma de la gaussienne.

N = ceil(3*sigma);
[x,y] = meshgrid(-N:N);

Gx = -x .* exp(-
(x.^2+y.^2)/(2*sigma^2)) /
(2*pi*sigma^4);
```

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

```
Gy = -y .* exp(-
(x.^2+y.^2)/(2*sigma^2)) /
(2*pi*sigma^4);

Gradx = conv2(I, Gx, 'same');
Grady = conv2(I, Gy, 'same');

end
```

```
function D =
harrisDetect(Gradx,Grady,k,sigma)

% Summary : Creation du detecteur de
Harris a partir du gradient
% d'intensite d'une image
%
% Description : Une fois le gradient
d'intensite obtenu a travers Gradx et
% Grady on utilise une autre
gaussienne G pour determiner les
coefficients
% de la matrice de covariance. On peut
donc encore parametrer la gaussienne
% grace a sigma. On calcul ensuite le
detecteur de Harris D, parametre par
% k.

N = ceil(3*sigma);
[x,y]=meshgrid(-N:N);

G=exp(-
(x.^2+y.^2)/(2*sigma^2))/(2*pi*sigma*s
igma);

A=Gradx.^2;
B=Gradx.*Grady;
C=Grady.^2;

a=conv2(A, G, 'same');
b=conv2(B, G, 'same');
c=conv2(C, G, 'same');

D=a.*c-b.^2-k*(a+c).^2;

end
```

```
function [hG,hD,bG,bD] =
zoneCoin(P,delta)

% Summary : On determine un carre
centre autour d'un point
%
% Description : On donne les quatre
coins d'un carre centre autour d'un
% point P. La taille du carre est de
2*delta.

R1=round(P(1));
R2=round(P(2));

hG = [R1-delta  R2-delta];
hD = [R1+delta  R2-delta];
bG = [R1-delta  R2+delta];
bD = [R1+delta  R2+delta];

end
```

```
function Pmax = maxCoin(D,hG,hD,bG,bD)

% Summary : Recherche d'un maximum
dans une zone restreinte d'une matrice
%
% Description : On prend en arguments
les quatre coins d'un rectangle/carre
% et on recherche le maximum de ce
quadrilatere au sein d'une matrice D.

Imax=0;

for i=hG(2):bG(2)
    for j=hG(1):hD(1)
        if D(i,j)>Imax
            imax=i;
            jmax=j;
            Imax=D(i,j);
        end
    end
end

Pmax=[jmax imax];

end
```

```
%% Test detection de Harris

clear
clc
close all

% Creation de la luminance I de la
frame 150 de la video
v = VideoReader('vid_in2.mp4');
ImRGB=read(v,150);
ImYCbCr=rgb2ycbcr(ImRGB);
I=double(ImYCbCr(:,:,1));


tic

[Ix,Iy]=intensityGradient(I,2);

D1=harrisDetect(Ix,Iy,0.05,3);
D2=harrisDetect(Ix,Iy,0.05,5);

A1=D1>0;
A2=D2>0;

D=A1.*A2.*D1.*D2; % Multiplication de
deux detecteurs

toc


figure

subplot(3,1,1)
imshow(0.5+Ix*0.3)

subplot(3,1,2)
imshow(0.5+Iy*0.3)

subplot(3,1,3)
imshow(D)
```

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

```
%% Creation de la video finale

clear
clc
close all

tic


newV = VideoWriter('video.avi');
newV.FrameRate = 25;
open(newV);

Vid = VideoReader('vid_in2.mp4');
ImRGB=readFrame(Vid);
nI1=reshape(double(ImRGB(:,:,1)),1,[])
;
nI2=reshape(double(ImRGB(:,:,2)),1,[])
;
nI3=reshape(double(ImRGB(:,:,3)),1,[])
;
ImYCbCr=rgb2ycbcr(ImRGB);
I=double(ImYCbCr(:,:,1));


imshow(ImRGB);
[x,y] = getpts;
P1=[x,y];
[x,y] = getpts;
P2=[x,y];
[x,y] = getpts;
P3=[x,y];
[x,y] = getpts;
P4=[x,y];
[x,y] = getpts;
P5=[x,y];
[x,y] = getpts;
P6=[x,y];


k=0;

while hasFrame(Vid)

    [hG1,hD1,bG1,bD1] =
zoneCoin(P1,25);
    [hG2,hD2,bG2,bD2] =
zoneCoin(P2,25);
    [hG3,hD3,bG3,bD3] =
zoneCoin(P3,25);
    [hG4,hD4,bG4,bD4] =
zoneCoin(P4,25);
    [hG5,hD5,bG5,bD5] =
zoneCoin(P5,25);
    [hG6,hD6,bG6,bD6] =
zoneCoin(P6,25);

    [Ix,Iy]=intensityGradient(I,2);

    D1=harrisDetect(Ix,Iy,0.05,3);
    D2=harrisDetect(Ix,Iy,0.05,5);

    A1=D1>0;
    A2=D2>0;

    D=A1.*A2.*D1.*D2;

    P1 = maxCoin(D,hG1,hD1,bG1,bD1);
    P2 = maxCoin(D,hG2,hD2,bG2,bD2);
    P3 = maxCoin(D,hG3,hD3,bG3,bD3);
    P4 = maxCoin(D,hG4,hD4,bG4,bD4);
    P5 = maxCoin(D,hG5,hD5,bG5,bD5);
    P6 = maxCoin(D,hG6,hD6,bG6,bD6);


    k=k+1

    A =
double(imread('cameraman.tif'));

    [x_max,y_max,P] = infoImage(A);
    H = homographie(P,P1,P2,P3,P4);


    x = min([P1(1) P2(1) P3(1)
P4(1)]):max([P1(1) P2(1) P3(1)
P4(1)]);
    y = min([P1(2) P2(2) P3(2)
P4(2)]):max([P1(2) P2(2) P3(2)
P4(2)]);


I1=reshape(double(ImRGB(y,x,1)),1,[]);

I2=reshape(double(ImRGB(y,x,2)),1,[]);

I3=reshape(double(ImRGB(y,x,3)),1,[]);


    n=length(x);
    m=length(y);

    [X,Y]=meshgrid(x,y);

    vY=reshape(Y,1,m*n);
    vX=reshape(X,1,m*n);
    vI=ones(1,m*n);

    U=[vX;vY;vI];
    V=H*U;

VN=[V(1,:)./V(3,:);V(2,:)./V(3,:)];



t=(VN(1,:)>=1).*(VN(1,:)<x_max).*(VN(2
,:)>=1).*(VN(2,:)<y_max);

tM=(VN(1,:)<=x_max).*(VN(1,:)>=3*x_max
/4).*(VN(2,:)>=y_max/4).*(VN(2,:)<=3*y
_max/4);

tI=(I1>=87).*(I1<=135).*(I2>=117).*(I3
>117);

    s1=t+tM;
    s2=t+tM+tI;

    ind1=find(s1==1);
    ind2=find(s2==3);


    U1=U(:,ind1);
    U2=U(:,ind2);
    VN1=VN(:,ind1);
    VN2=VN(:,ind2);


    nI1(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);
    nI2(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);
```

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

```
    nI3(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);

    nI1(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);
    nI2(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);
    nI3(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);


    nI1=reshape(nI1,1080,1920);
    nI2=reshape(nI2,1080,1920);
    nI3=reshape(nI3,1080,1920);

    nImRGB=cat(3,nI1,nI2,nI3);


    [P,M] =
projection(P1,P2,P3,P4,P5,P6);

    S0 =
conv2Dnorm([0.375;0.375;3;1],P);
    S1 = conv2Dnorm([0;0.5;0;1],P);
    S2 = conv2Dnorm([0.375;0;0;1],P);
    S3 = conv2Dnorm([0.75;0.5;0;1],P);
    S4 = conv2Dnorm([0.375;1;0;1],P);

    nImRGB = insertPyr(nImRGB,
S0,S1,S2,S3,S4);


    writeVideo(newV, uint8(nImRGB));

    ImRGB=readFrame(Vid);

nI1=reshape(double(ImRGB(:,:,1)),1,[])
;

nI2=reshape(double(ImRGB(:,:,2)),1,[])
;

nI3=reshape(double(ImRGB(:,:,3)),1,[])
;
    ImYCbCr=rgb2ycbcr(ImRGB);
    I=double(ImYCbCr(:,:,1));

end
```