LIÈGE université Library

http://lib.uliege.be https://matheo.uliege.be

Création d'un Modèle Numérique Dynamique de Surface (MNDS) à l'aide de l'algorithme de la Direct Linear Transformation (DLT) et détermination des champs de vitesses de l'écoulement à partir d'appareils vidéographiques

Auteur : Decalf, Manon
Promoteur(s) : Cornet, Yves; Dewals, Benjamin
Faculté : Faculté des Sciences
Diplôme : Master en sciences géographiques, orientation géomatique et géométrologie, à finalité spécialisée
Année académique : 2017-2018
URI/URL : http://hdl.handle.net/2268.2/5584

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

UNIVERSITÉ DE LIÈGE
Faculté des Sciences
Département de Géographie

Création d'un Modèle Numérique Dynamique de Surface (MNDS) à l'aide de l'algorithme de la *Direct Linear Transformation* (DLT) et détermination des champs de vitesses de l'écoulement à partir d'appareils vidéographiques

---

ANNEXES

Date de défense : Septembre 2018
Promoteur : Y. CORNET
Co-promoteur : B. DEWALS
Lecteurs : P. ARCHAMBEAU & G. HOUBRECHTS

Mémoire présenté par :
Manon DECALF
pour l'obtention du titre de :
Master en Sciences géographiques, orientation géomatique et géométrologie, à finalité spécialisée

Annexes numériques

Les annexes numériques sont composées du script de profilométrie laser développé par Rifai (2016) au sein duquel les fonctions et les données relatives à ce mémoire ont été rajoutées.

Scripts rajoutés au code initial:

- photogrammetrie.m (script principal) (script 6)
- classification_flotteur.m (script 1)
- erreur_coplanarite_jaune.m (script 3)
- erreur_coplanarite_orange.m (script 4)
- geometrie_epipolaire.m (script 2)
- DLT_Restitution_16p_modif.m (script 5)

Images :

- Les trois séquences vidéo sont disponibles sur demande.

Données :

- GCPmonde.mat (coordonnées-monde des PV)
- hom_Canon.mat (coordonnées-image des PV)
- hom_Lumix72.mat (coordonnées-image des PV)
- hom_LumixGH4.mat (coordonnées-image des PV)
- Points_control.mat (coordonnées-monde des PC)
- u_v_Canon.mat (coordonnées-image des PC)
- u_v_Lumix72.mat (coordonnées-image des PC)
- u_v_LumixGH4.mat (coordonnées-image des PC)

Script 1 : classification_flotteur.m

```
%% Lecture image

image_originale = image_rgb_full(:,:,:);
red = image_originale(:,:,1);
green = image_originale(:,:,2);
blue = image_originale(:,:,3);

%% Masque binaire de la zone eau

while choix == 0
    [masque_zone,x,y] = roipoly(image_originale);
    choix = questdlg('Valider le masque ?', 'Masque','Oui', 'Non', 'Non');
    close();
    imshow(masque_zone);

    switch choix
        case 'Oui'
            choix = 1;
        case 'Non'
            choix = 0;
    end
end

masque_zone = uint8(masque_zone);

maskedRed = red.* masque_zone;
maskedGreen = green.* masque_zone;
maskedBlue = blue.* masque_zone;

maskedRgbImage = cat(3, maskedRed, maskedGreen, maskedBlue);
imshow(maskedRgbImage);

%% Détection des flotteurs

hsv = rgb2hsv(maskedRgbImage);

h = hsv(:,:,1); %teinte
s = hsv(:,:,2); %saturation
v = hsv(:,:,3); %intensité

switch id_fold

    case 1
        flotteur_jaune = ((h >= 0.131) & (h <= 0.241)) & ((s >= 0.296) & (s <= 1.0)) & ((v >= 0.391) & (v <= 1.0));
        flotteur_orange = ((h >= 0.943) | (h <= 0.036)) & ((s >= 0.354) & (s <= 1.0)) & ((v >= 0.346) & (v <= 1.0));
    case 2
        flotteur_jaune = ((h >= 0.086) & (h <= 0.259)) & ((s >= 0.272) & (s <= 1.0)) & ((v >= 0.458 ) & (v <= 1.0));
        flotteur_orange = ((h >= 0.946 ) | (h<= 0.048)) & ((s >= 0.485) & (s <= 1.0)) & ((v >= 0.313) & (v <= 1.0));
    case 3
        flotteur_jaune = ((h >= 0.110) & (h <= 0.251)) & ((s >= 0.322 ) & (s <= 1.0)) & ((v >= 0.208) & (v <= 1.0));
        flotteur_orange = ((h >= 0.940) | (h<= 0.056)) & ((s >= 0.350 ) & (s <= 1.0)) & ((v >= 0.106) & (v <= 1.0));
end

imshow(flotteur_jaune);
imshow(flotteur_orange);

%% Erosion et dilatation

SE_erosion = strel('disk',1,8);
SE = strel('disk',1,8);

%Flotteurs oranges
erosion_orange = imerode(flotteur_orange,SE_erosion);
imshow(erosion_orange);
dilatation_orange = imdilate(erosion_orange,SE);
imshow(dilatation_orange);

%Flotteurs jaunes
erosion_jaune = imerode(flotteur_jaune,SE_erosion);
imshow(erosion_jaune);
dilatation_jaune = imdilate(erosion_jaune,SE);
imshow(dilatation_jaune);

%% Labelling et centroid de chaque flotteur

%Flottteurs oranges
[L_orange,n_orange]= bwlabel(dilatation_orange);

orange = im2bw(dilatation_orange);
orange = uint8(orange);
maskedRed_orange = maskedRed.* orange;
maskedGreen_orange = maskedGreen.* orange;
maskedBlue_orange = maskedBlue.* orange;
maskedRgbImage_orange = cat(3, maskedRed_orange, maskedGreen_orange, maskedBlue_orange);
imshow(maskedRgbImage_orange);

image_gris_orange = rgb2gray(maskedRgbImage_orange);
grey_orange = regionprops(dilatation_orange,image_gris_orange,'WeightedCentroid','Extrema','area','Eccentricity');
data_grey_orange = struct2table(grey_orange);
nbr_orange = size(data_grey_orange);
nbr_orange = n_orange(1,1);

%Flotteurs jaunes
[L_jaune,n_jaune]= bwlabel(dilatation_jaune);

```

```
jaune = im2bw(dilatation_jaune);
jaune = uint8(jaune);
maskedRed_jaune = maskedRed.* jaune;
maskedGreen_jaune = maskedGreen.* jaune;
maskedBlue_jaune = maskedBlue.* jaune;
maskedRgbImage_jaune = cat(3, maskedRed_jaune, maskedGreen_jaune, maskedBlue_jaune);
imshow(maskedRgbImage_jaune);

image_gris_jaune = rgb2gray(maskedRgbImage_jaune);
grey_jaune = regionprops(dilatation_jaune,image_gris_jaune,'WeightedCentroid','Extrema','area','Eccentricity');
data_grey_jaune = struct2table(grey_jaune);
nbr_jaune = size(data_grey_jaune);
nbr_jaune = n_jaune(1,1);

%% Cas particuliers

% Flotteurs jaunes
for i = 1:nbr_jaune
    if((data_grey_jaune.Area(i) >= 40) && (data_grey_jaune.Eccentricity(i) >= 0.80))

        y_classif_min = round(data_grey_jaune.Extrema{i}(1,2));
        y_classif_max = round(data_grey_jaune.Extrema{i}(6,2));
        x_centroid = round(data_grey_jaune.WeightedCentroid(i,1));
        delta_y = abs(y_classif_max - y_classif_min);

        x_classif_max = round(data_grey_jaune.Extrema{i}(3,1));
        x_classif_min = round(data_grey_jaune.Extrema{i}(8,1));
        y_centroid = round(data_grey_jaune.WeightedCentroid(i,2));
        delta_x = abs(x_classif_max - x_classif_min);

        if (delta_y > delta_x)

            for j = x_classif_min:x_classif_max
                L_jaune(y_centroid,j) = 0;
            end

        else

            for j = y_classif_min:y_classif_max
                L_jaune(j,x_centroid) = 0;
            end
        end
        imshow(L_jaune);
        erosion_jaune = imerode(L_jaune,SE_erosion);
        imshow(erosion_jaune);
        dilatation_jaune = imdilate(erosion_jaune,SE);
        imshow(dilatation_jaune);
    end
end

[L_jaune,n_jaune]= bwlabel(dilatation_jaune);
properties_cas_jaune = regionprops(L_jaune,image_gris_jaune,'Extrema','area','Eccentricity','WeightedCentroid');
data_grey_jaune = struct2table(properties_cas_jaune);

%Flotteurs oranges
for i = 1:nbr_orange
    if((data_grey_orange.Area(i) >= 40) && (data_grey_orange.Eccentricity(i) >= 0.83))

        y_classif_min = round(data_grey_orange.Extrema{i}(1,2));
        y_classif_max = round(data_grey_orange.Extrema{i}(6,2));
        x_centroid = round(data_grey_orange.WeightedCentroid(i,1));
        delta_y = abs(y_classif_max - y_classif_min);

        x_classif_max = round(data_grey_orange.Extrema{i}(3,1));
        x_classif_min = round(data_grey_orange.Extrema{i}(8,1));
        y_centroid = round(data_grey_orange.WeightedCentroid(i,2));
        delta_x = abs(x_classif_max - x_classif_min);

        if (delta_y > delta_x)

            for j = x_classif_min:x_classif_max
                L_orange(y_centroid,j) = 0;
            end

        else

            for j = y_classif_min:y_classif_max
                L_orange(j,x_centroid) = 0;
            end
        end

        imshow(L_orange);
        erosion_orange = imerode(L_orange,SE_erosion);
        imshow(erosion_orange);
        dilatation_orange = imdilate(erosion_orange,SE);
        imshow(dilatation_orange);
    end
end

[L_orange,n_orange] = bwlabel(dilatation_orange);
properties_cas_orange = regionprops(L_orange,image_gris_orange,'Extrema','area','Eccentricity','WeightedCentroid');
data_grey_orange = struct2table(properties_cas_orange);
```

Script 2 : geometrie_epipolaire.m

```
clearvars x_intersection1 y_intersection1 x_intersection2 y_intersection2

switch camera2

    case 1
        x_capteur = 36;
        y_capteur = 24;
        x_resolution_max = 5760;
        y_resolution_max = 3840;
        load Param_DLT_Canon.mat;
    case 2
        x_capteur = 6.16;
        y_capteur = 4.62;
        x_resolution_max = 4608;
        y_resolution_max = 3456;
        load Param_DLT_Lumix72.mat;
    case 3
        x_capteur = 17.3;
        y_capteur = 13;
        x_resolution_max = 4608;
        y_resolution_max = 3456;
        load Param_DLT_LumixGH4.mat;
end

%% 1er étape : PLAN BASE-NI en coordonnée monde

fprintf('1.1 Vecteur NI en coordonnée image 1\n\n');
NI1_I(1,1) = x_centroid - PPuo_stock(camera1,1);
NI1_I(1,2) = (Y_px - y_centroid) - PPvo_stock(camera1,1);
NI1_I(1,3) = Z_stock(camera1,1);

fprintf('1.2 Vecteur NI en coordonnée monde\n\n');
T1_M_I = T_stock((camera1 + 2*(camera1-1)):(camera1 + (2*(camera1-1))+2),1:3);
NI_M(1,:) = T1_M_I * NI1_I(1,:)';

fprintf('1.3 Vecteur BASE en coordonnée monde\n\n');
B_M(1:3,1) = posiCam_stock(:,camera2) - posiCam_stock(:,camera1);

Plan_base(1,1:3) = cross(B_M(1:3,1)',NI_M(1,:));

fprintf('1.4 Equation du plan BASE-NI en coordonnée monde \n\n');
syms d
eqn1 = Plan_base(1,1)*posiCam_stock(1,camera2) + Plan_base(1,2)*posiCam_stock(2,camera2) + Plan_base(1,3)*posiCam_stock(3,camera2) + d == 0;
d_plan_base(1,1) = solve(eqn1,d);

%% 2ème étape : PLAN IMAGE 2 en coordonnée monde

fprintf('2.1 Vecteur normal à l image 2 en coordonnée monde\n\n');
N2_PP2_image(1,1) = 0;
N2_PP2_image(2,1) = 0;
N2_PP2_image(3,1) = Z_stock(camera2,1);

T2_M_I = T_stock((camera2 + 2*(camera2-1)):(camera2 + (2*(camera2-1))+2),1:3);
N2_PP2_monde(1:3,1) = T2_M_I * N2_PP2_image(:,1);

fprintf('2.2 Position du point principal en coordonnée monde\n\n');
lambda_u = (x_capteur/x_resolution_max)*0.001;%4896
lambda_v = (y_capteur/y_resolution_max)*0.001;%3672
lambda_mean = (lambda_u + lambda_v)/2;
focale = -1 * lambda_mean * Z_stock(camera2,1);%focale approximative%0.00375mm

PP2_monde = zeros(3,1);
PP2_monde(1,1) = posiCam_stock(1,camera2) + (N2_PP2_monde(1,1)/sqrt(N2_PP2_monde(1,1)^2 + N2_PP2_monde(2,1)^2 + N2_PP2_monde(3,1)^2))*focale;
PP2_monde(2,1) = posiCam_stock(2,camera2) + (N2_PP2_monde(2,1)/sqrt(N2_PP2_monde(1,1)^2 + N2_PP2_monde(2,1)^2 + N2_PP2_monde(3,1)^2))*focale;
PP2_monde(3,1) = posiCam_stock(3,camera2) + (N2_PP2_monde(3,1)/sqrt(N2_PP2_monde(1,1)^2 + N2_PP2_monde(2,1)^2 + N2_PP2_monde(3,1)^2))*focale;

fprintf('2.3 Equation du plan IMAGE 2 en coordonnée monde\n\n');
syms d_2
eqn2 = N2_PP2_monde(1,1)*PP2_monde(1,1) + N2_PP2_monde(2,1)*PP2_monde(2,1) + N2_PP2_monde(3,1)*PP2_monde(3,1) + d_2 == 0;
plan_image(1,1) = solve(eqn2,d_2);

%% 3ème étape : Equation de la droite épipolaire

%3.1 : Résolution du système
syms x y z
f1 = Plan_base(1,1)*x + Plan_base(1,2)*y + Plan_base(1,3)*z + d_plan_base(1,1) == 0;
f2 = N2_PP2_monde(1,1)*x + N2_PP2_monde(2,1)*y + N2_PP2_monde(3,1)*z + plan_image(1,1) == 0;
S = solve(f1,f2);

%% 4ème étape : Intersection de la droite épipolaire avec les bords de l'image 2 dans le référentiel-monde

%4.1 : Vecteurs directeurs des bords de l'image référentiel image 2
v4(1:3,1) = [X_px;0;0];
v3(1:3,1) = [0;Y_px;0];
v2(1:3,1) = [X_px;0;0];
v1(1:3,1) = [0;Y_px;0];

%4.2 : Transfo pour obtenir vecteur directeur des bords de l'image en système monde
v1_monde(1:3,1) = T2_M_I * v1(1:3,1);
v2_monde(1:3,1) = T2_M_I * v2(1:3,1);
v3_monde(1:3,1) = T2_M_I * v3(1:3,1);
v4_monde(1:3,1) = T2_M_I * v4(1:3,1);

%4.3 : Coordonnées des coins de l'image dans le système monde
```

```
    c1_PP2_dist_image = sqrt(PPuo_stock(camera2,1)^2+PPvo_stock(camera2,1)^2);%distance image coins - point principal
    c3_PP2_dist_image = sqrt((X_px-PPuo_stock(camera2,1))^2+((Y_px-PPvo_stock(camera2,1))^2));
    c2_PP2_dist_image = sqrt(((Y_px-PPvo_stock(camera2,1))^2)+PPuo_stock(camera2,1)^2);
    c4_PP2_dist_image = sqrt(((X_px-PPuo_stock(camera2,1))^2)+PPvo_stock(camera2,1)^2);

    c1_N2_dist_image = sqrt((c1_PP2_dist_image^2)+(Z_stock(camera2,1)^2));%distance image coins - centre de projection
    c3_N2_dist_image = sqrt((c3_PP2_dist_image^2)+(Z_stock(camera2,1)^2));
    c2_N2_dist_image = sqrt((c2_PP2_dist_image^2)+(Z_stock(camera2,1)^2));
    c4_N2_dist_image = sqrt((c4_PP2_dist_image^2)+(Z_stock(camera2,1)^2));

    c1_N2_dist_monde = lambda_mean * (c1_N2_dist_image);%distance monde coins - centre de projection
    c3_N2_dist_monde = lambda_mean * (c3_N2_dist_image);
    c2_N2_dist_monde = lambda_mean * (c2_N2_dist_image);
    c4_N2_dist_monde = lambda_mean * (c4_N2_dist_image);

    c1_N2_image(1:3,1) = [-1*PPuo_stock(camera2,1); -1*PPvo_stock(camera2,1); Z_stock(camera2,1)];%Vecteur directeur coins image-centre de projection
    c3_N2_image(1:3,1) = [X_px - PPuo_stock(camera2,1);Y_px - PPvo_stock(camera2,1); Z_stock(camera2,1)];
    c2_N2_image(1:3,1) = [-1* PPuo_stock(camera2,1);Y_px - PPvo_stock(camera2,1); Z_stock(camera2,1)];
    c4_N2_image(1:3,1) = [X_px - PPuo_stock(camera2,1); -1*PPvo_stock(camera2,1); Z_stock(camera2,1)];

    c1_N2_monde(1:3,1) = T2_M_I * c1_N2_image(1:3,1);%Vecteur directeur coins image-centre de projection en système monde
    c3_N2_monde(1:3,1) = T2_M_I * c3_N2_image(1:3,1);
    c2_N2_monde(1:3,1) = T2_M_I * c2_N2_image(1:3,1);
    c4_N2_monde(1:3,1) = T2_M_I * c4_N2_image(1:3,1);

    Point_monde_1 = zeros(3,1);%-0.013
    Point_monde_1(1,1) = posiCam_stock(1,camera2) + (c1_N2_monde(1,1)/sqrt(c1_N2_monde(1,1)^2 + c1_N2_monde(2,1)^2 + ...
        c1_N2_monde(3,1)^2))*c1_N2_dist_monde;
    Point_monde_1(2,1) = posiCam_stock(2,camera2) + (c1_N2_monde(2,1)/sqrt(c1_N2_monde(1,1)^2 + c1_N2_monde(2,1)^2 + ...
        c1_N2_monde(3,1)^2))*c1_N2_dist_monde;
    Point_monde_1(3,1) = posiCam_stock(3,camera2) + (c1_N2_monde(3,1)/sqrt(c1_N2_monde(1,1)^2 + c1_N2_monde(2,1)^2 + ...
        c1_N2_monde(3,1)^2))*c1_N2_dist_monde;

    Point_monde_3 = zeros(3,1);%0.013
    Point_monde_3(1,1) = posiCam_stock(1,camera2) + (c3_N2_monde(1,1)/sqrt(c3_N2_monde(1,1)^2 + c3_N2_monde(2,1)^2 + ...
        c3_N2_monde(3,1)^2))*c3_N2_dist_monde;
    Point_monde_3(2,1) = posiCam_stock(2,camera2) + (c3_N2_monde(2,1)/sqrt(c3_N2_monde(1,1)^2 + c3_N2_monde(2,1)^2 + ...
        c3_N2_monde(3,1)^2))*c3_N2_dist_monde;
    Point_monde_3(3,1) = posiCam_stock(3,camera2) + (c3_N2_monde(3,1)/sqrt(c3_N2_monde(1,1)^2 + c3_N2_monde(2,1)^2 + ...
        c3_N2_monde(3,1)^2))*c3_N2_dist_monde;

    Point_monde_2 = zeros(3,1);%0.013
    Point_monde_2(1,1) = posiCam_stock(1,camera2) + (c2_N2_monde(1,1)/sqrt(c2_N2_monde(1,1)^2 + c2_N2_monde(2,1)^2 + ...
        c2_N2_monde(3,1)^2))*c2_N2_dist_monde;
    Point_monde_2(2,1) = posiCam_stock(2,camera2) + (c2_N2_monde(2,1)/sqrt(c2_N2_monde(1,1)^2 + c2_N2_monde(2,1)^2 + ...
        c2_N2_monde(3,1)^2))*c2_N2_dist_monde;
    Point_monde_2(3,1) = posiCam_stock(3,camera2) + (c2_N2_monde(3,1)/sqrt(c2_N2_monde(1,1)^2 + c2_N2_monde(2,1)^2 + ...
        c2_N2_monde(3,1)^2))*c2_N2_dist_monde;

    Point_monde_4 = zeros(3,1);%0.013
    Point_monde_4(1,1) = posiCam_stock(1,camera2) + (c4_N2_monde(1,1)/sqrt(c4_N2_monde(1,1)^2 + c4_N2_monde(2,1)^2 + ...
        c4_N2_monde(3,1)^2))*c4_N2_dist_monde;
    Point_monde_4(2,1) = posiCam_stock(2,camera2) + (c4_N2_monde(2,1)/sqrt(c4_N2_monde(1,1)^2 + c4_N2_monde(2,1)^2 + ...
        c4_N2_monde(3,1)^2))*c4_N2_dist_monde;
    Point_monde_4(3,1) = posiCam_stock(3,camera2) + (c4_N2_monde(3,1)/sqrt(c4_N2_monde(1,1)^2 + c4_N2_monde(2,1)^2 + ...
        c4_N2_monde(3,1)^2))*c4_N2_dist_monde;

    %4.4 : Vérification des coordonnées monde obtenus pour les coins
    dist_v4 = sqrt((Point_monde_1(1,1)-Point_monde_4(1,1))^2+(Point_monde_1(2,1)-Point_monde_4(2,1))^2+(Point_monde_1(3,1)-Point_monde_4(3,1))^2)
    dist_v1 = sqrt((Point_monde_1(1,1)-Point_monde_2(1,1))^2+(Point_monde_1(2,1)-Point_monde_2(2,1))^2+(Point_monde_1(3,1)-Point_monde_2(3,1))^2)
    dist_v2 = sqrt((Point_monde_2(1,1)-Point_monde_3(1,1))^2+(Point_monde_2(2,1)-Point_monde_3(2,1))^2+(Point_monde_2(3,1)-Point_monde_3(3,1))^2)
    dist_v3 = sqrt((Point_monde_3(1,1)-Point_monde_4(1,1))^2+(Point_monde_3(2,1)-Point_monde_4(2,1))^2+(Point_monde_3(3,1)-Point_monde_4(3,1))^2)

    dist_v1_px = dist_v1/lambda_mean;
    dist_v2_px = dist_v2/lambda_mean;
    dist_v3_px = dist_v3/lambda_mean;
    dist_v4_px = dist_v4/lambda_mean;

    %4.5 : Réoslution du système en coordonnée monde
    syms x y z t
    f11 = S.x == x;
    f21 = S.y == y;
    f31 = t*v1_monde(3,1) + Point_monde_1(3,1) == z;
    f41 = t*v1_monde(1,1) + Point_monde_1(1,1) == x;
    reso_v1 = solve(f11,f21,f31,f41);

    syms x y z t
    f12 = S.x == x;
    f22 = S.y == y;
    f32 = t*v2_monde(3,1) + Point_monde_2(3,1) == z;
    f42 = t*v2_monde(1,1) + Point_monde_2(1,1) == x;
    reso_v2 = solve(f12,f22,f32,f42);

    syms x y z t
    f13 = S.x == x;
    f23 = S.y == y;
    f33 = t*v3_monde(3,1) + Point_monde_3(3,1) == z;
    f43 = t*v3_monde(1,1) + Point_monde_3(1,1) == x;
    reso_v3 = solve(f13,f23,f33,f43);

    syms x y z t
    f14 = S.x == x;
    f24 = S.y == y;
    f34 = t*v4_monde(3,1) + Point_monde_4(3,1) == z;
    f44 = t*v4_monde(1,1) + Point_monde_4(1,1) == x;
    reso_v4 = solve(f14,f24,f34,f44);
```

```
%% 5ème étape : transformation des coordonnées des points d'intersection en coordonnée image 2

u_v1_image = ((L(1,1)*reso_v1.x) + (L(2,1)*reso_v1.y) + (L(3,1)*reso_v1.z) + L(4,1))/((L(9,1)*reso_v1.x) + (L(10,1)*reso_v1.y) + ...
    (L(11,1)*reso_v1.z)+1);
v_v1_image = ((L(5,1)*reso_v1.x) + (L(6,1)*reso_v1.y) + (L(7,1)*reso_v1.z) + L(8,1))/((L(9,1)*reso_v1.x) + (L(10,1)*reso_v1.y) + ...
    (L(11,1)*reso_v1.z)+1);

u_v2_image = ((L(1,1)*reso_v2.x) + (L(2,1)*reso_v2.y) + (L(3,1)*reso_v2.z) + L(4,1))/((L(9,1)*reso_v2.x) + (L(10,1)*reso_v2.y) + ...
    (L(11,1)*reso_v2.z)+1);
v_v2_image = ((L(5,1)*reso_v2.x) + (L(6,1)*reso_v2.y) + (L(7,1)*reso_v2.z) + L(8,1))/((L(9,1)*reso_v2.x) + (L(10,1)*reso_v2.y) + ...
    (L(11,1)*reso_v2.z)+1);

u_v3_image = ((L(1,1)*reso_v3.x) + (L(2,1)*reso_v3.y) + (L(3,1)*reso_v3.z) + L(4,1))/((L(9,1)*reso_v3.x) + (L(10,1)*reso_v3.y) + ...
    (L(11,1)*reso_v3.z)+1);
v_v3_image = ((L(5,1)*reso_v3.x) + (L(6,1)*reso_v3.y) + (L(7,1)*reso_v3.z) + L(8,1))/((L(9,1)*reso_v3.x) + (L(10,1)*reso_v3.y) + ...
    (L(11,1)*reso_v3.z)+1);

u_v4_image = ((L(1,1)*reso_v4.x) + (L(2,1)*reso_v4.y) + (L(3,1)*reso_v4.z) + L(4,1))/((L(9,1)*reso_v4.x) + (L(10,1)*reso_v4.y) + ...
    (L(11,1)*reso_v4.z)+1);
v_v4_image = ((L(5,1)*reso_v4.x) + (L(6,1)*reso_v4.y) + (L(7,1)*reso_v4.z) + L(8,1))/((L(9,1)*reso_v4.x) + (L(10,1)*reso_v4.y) + ...
    (L(11,1)*reso_v4.z)+1);

bord(1,1) = double(u_v1_image);
bord(1,2) = double(v_v1_image);
bord(2,1) = double(u_v2_image);
bord(2,2) = double(v_v2_image);
bord(3,1) = double(u_v3_image);
bord(3,2) = double(v_v3_image);
bord(4,1) = double(u_v4_image);
bord(4,2) = double(v_v4_image);

for i=1:4
    if(((-5)<=round(double(bord(i,1))))&&(round(double(bord(i,1)))<=(X_px+5))&&((-5)<=round(double(bord(i,2))))&&...
            (round(double(bord(i,2)))<=(Y_px+5)))
        x_intersection1 = bord(i,1);
        y_intersection1 = bord(i,2);
        for j=(i+1):4
            if(((-5)<=round(double(bord(j,1))))&&(round(double(bord(j,1)))<=(X_px+5))&&((-5)<=round(double(bord(j,2))))&&...
                    (round(double(bord(j,2)))<=(Y_px+5)))
                x_intersection2 = bord(j,1);
                y_intersection2 = bord(j,2);
                for k=j:4
                    bord(k,:)=-6;
                end
            end
        end
    end
end

pente = (y_intersection2 - y_intersection1)/(x_intersection2 - x_intersection1);
intercept = y_intersection2 - (pente * x_intersection2);
```

Script 3 : erreur_coplanarite_jaune.m

```
clear produit_vectoriel produit_scalaire

%Rayon 1 monde
rayon1_image(1,1) = homologue_jaune(w).lumix72{h,1} - PPuo_stock(camera1,1);
rayon1_image(2,1) = (Y_px - homologue_jaune(w).lumix72{h,2}) - PPvo_stock(camera1,1);
rayon1_image(3,1) = Z_stock(camera1,1);

T_rayon1_M_I = T_stock((camera1 + 2*(camera1-1)):(camera1 + (2*(camera1-1))+2),1:3);
rayon1_monde(:,1) = T_rayon1_M_I * rayon1_image(:,1);

%Composantes monde BASE
Base_M(1:3,1) = posiCam_stock(:,camera2) - posiCam_stock(:,camera1);

%Rayon 2 monde
for g = 1:nbr_candidat_jaune

    ind = candidat_jaune(w).proche_inter_indice{h,g};
    rayon2_image(1,g) = Canon_data_jaune(w).flotteur{ind,1} - PPuo_stock(camera2,1);
    rayon2_image(2,g) = (Y_px - Canon_data_jaune(w).flotteur{ind,2}) - PPvo_stock(camera2,1);
    rayon2_image(3,g) = Z_stock(camera2,1);

    T_rayon2_M_I = T_stock((camera2 + 2*(camera2-1)):(camera2 + (2*(camera2-1))+2),1:3);
    rayon2_monde(:,g) = T_rayon2_M_I * rayon2_image(:,g);

    %Erreur de coplanéité
    produit_vectoriel(:,g) = cross(rayon1_monde(:,1),rayon2_monde(:,g));
    produit_scalaire(:,g) = abs(dot(produit_vectoriel(:,g),Base_M(:,1)));

end
```

Script 4 : erreur_coplanarite_orange.m

```
clear produit_vectoriel produit_scalaire

%Rayon 1 monde
rayon1_image(1,1) = homologue_orange(w).lumix72{h,1} - PPuo_stock(camera1,1);
rayon1_image(2,1) = (Y_px - homologue_orange(w).lumix72{h,2}) - PPvo_stock(camera1,1);
rayon1_image(3,1) = Z_stock(camera1,1);

T_rayon1_M_I = T_stock((camera1 + 2*(camera1-1)):(camera1 + (2*(camera1-1))+2),1:3);
rayon1_monde(:,1) = T_rayon1_M_I * rayon1_image(:,1);

%Composantes monde BASE
Base_M(1:3,1) = posiCam_stock(:,camera2) - posiCam_stock(:,camera1);

%Rayon 2 monde
for r = 1:nbr_candidat_orange

    ind = candidat_orange(w).proche_inter_indice{h,r};
    rayon2_image(1,r) = Canon_data_orange(w).flotteur{ind,1} - PPuo_stock(camera2,1);
    rayon2_image(2,r) = (Y_px - Canon_data_orange(w).flotteur{ind,2}) - PPvo_stock(camera2,1);
    rayon2_image(3,r) = Z_stock(camera2,1);

    T_rayon2_M_I = T_stock((camera2 + 2*(camera2-1)):(camera2 + (2*(camera2-1))+2),1:3);
    rayon2_monde(:,r) = T_rayon2_M_I * rayon2_image(:,r);

    %Erreur de coplanéité
    produit_vectoriel(:,r) = cross(rayon1_monde(:,1),rayon2_monde(:,r));
    produit_scalaire(:,r) = abs(dot(produit_vectoriel(:,r),Base_M(:,1)));

end
```

Script 5 : DLT_Restitution_16p_modif.m

```
%% Restitution

%Allocation de la mémoire
xi=zeros(nHo,1,nCam);
eta=zeros(nHo,1,nCam);
rsquare=zeros(nHo,1,nCam);

ErrorDistoU=zeros(nHo,1,nCam);
ErrorDistoV=zeros(nHo,1,nCam);

muom=zeros(nHo,2,nCam);
muomSansDisto=zeros(nHo,2,nCam);

xyz=zeros(nHo,3);

%% Préparation des variables

%calcul des xi et eta et rsquare

for n=1:nCam


xi(:,1,n)=homologue(:,1,n)-PP(1,1,n);
eta(:,1,n)=homologue(:,2,n)-PP(1,2,n);%-Y_PX
rsquare(:,1,n)=xi(:,1,n).^2+ eta(:,1,n).^2;
```

```


%Calcul des distorsions radiales.

        for i=1:nHo

            ErrorDistoU(i,1,n)=xi(i,1,n)*(L_param(12,1,n)*rsquare(i,1,n)+L_param(13,1,n)*rsquare(i,1,n).^2+L_param(14,1,n)*rsquare(i,1,n).^3)+...
                L_param(15,1,n)*(rsquare(i,:,n)+2*xi(i,:,n).^2)+L_param(16,1,n)*xi(i,:,n)*eta(i,:,n);

            ErrorDistoV(i,1,n)=eta(i,1,n)*(L_param(12,1,n)*rsquare(i,1,n)+L_param(13,1,n)*rsquare(i,1,n).^2+L_param(14,1,n)*rsquare(i,1,n).^3)+...
                L_param(15,1,n)*xi(i,:,n)*eta(i,:,n)+L_param(16,1,n)*(rsquare(i,:,n)+2*eta(i,:,n).^2);

        end


muom(:,1,n)= homologue(:,1,n)-ErrorDistoU(:,1,n);
muom(:,2,n)= 1080-homologue(:,2,n)-ErrorDistoV(:,1,n);

muomSansDisto(:,1,n)=homologue(:,1,n)
muomSansDisto(:,2,n)=homologue(:,2,n)

end

%Réorganisation de la matrice xi eta (muom), pour des facilités de calcul.
```

```
muomb=reshape(muom,nHo,2*nCam);          %changer si on veut ou non pendre la disto en compte nuom ou muom sans disto.

for i=1:nHo

  if nCam<2


    disp('plus de deux caméras est nécéssaire')


  % Si il y a plus de deux caméras, on peut reconstruire.
  elseif nCam>=2

    cdx = 1:nCam

    m1=[];
    m2=[];

    m1(1:2:nCam*2,1)=muomb(i,(cdx*2)-1).*L_param(9,cdx)-L_param(1,cdx);
    m1(1:2:nCam*2,2)=muomb(i,(cdx*2)-1).*L_param(10,cdx)-L_param(2,cdx);
    m1(1:2:nCam*2,3)=muomb(i,(cdx*2)-1).*L_param(11,cdx)-L_param(3,cdx);

    m1(2:2:nCam*2,1)=muomb(i,cdx*2).*L_param(9,cdx)-L_param(5,cdx);
    m1(2:2:nCam*2,2)=muomb(i,cdx*2).*L_param(10,cdx)-L_param(6,cdx);
    m1(2:2:nCam*2,3)=muomb(i,cdx*2).*L_param(11,cdx)-L_param(7,cdx);


    m2(1:2:nCam*2,1)=L_param(4,cdx)-muomb(i,(cdx*2)-1);
    m2(2:2:nCam*2,1)=L_param(8,cdx)-muomb(i,cdx*2);


    xyz(i,1:3)=mldivide(m1,m2);

  end
end
```

Script 6 : photogrammetrie.m

```
%PHOTOGRAMMETRIE : MNS DYNAMIQUE
%DECALF MANON

clc
clear all
close all
format long

%% PARTIE 1 : Adaptation du code d'Ismail Rifai

main_images_directory = '..\Images\';
list_dir = dir(main_images_directory);
isub = [list_dir(:).isdir]; % returns logical vector
nameFolds = {list_dir(isub).name}';
nameFolds(ismember(nameFolds,{'.','..'})) = [];

%% affichage liste des dossiers image pour vérification si verification_des_fichiers = 1
verification_des_fichiers = 1;

if verification_des_fichiers
    old_folder = cd;
    for i = 1:size(nameFolds,1);
        im_dir = nameFolds{i};
        nb_image = size(ls([main_images_directory nameFolds{i} '\*.jpg']),1);
        fprintf(['dossier %d/%d : ' nameFolds{i} '   (nombre d''images : %d) \n'],i,size(nameFolds,1),nb_image)
    end
    pause()
end

%% Lien vers les routines
addpath('Sources\Routines');

%% PARAMETRAGE
% Choix des images à traiter
pas_image = 1;
derniere_image = 0;                                          % zero pour aller jusqu'à la dernière image
dossiers_a_traiter = 3;                                     % dossier image à traiter

posiCam_stock = zeros(3,dossiers_a_traiter);
T_stock = zeros(dossiers_a_traiter*3,3);
ypr_stock = zeros(dossiers_a_traiter,3);
PPuo_stock = zeros(dossiers_a_traiter,1);
PPvo_stock = zeros(dossiers_a_traiter,1);
Z_stock = zeros(dossiers_a_traiter,1);

%% Activation de la calibration
% si = 0 : chargement des fichier *.mat correspondants
% si = 1 : calibration et création des fichiers *.mat correspondants

detection_des_zones_digue_et_axes = 0;
```

```
detection_des_zones_eau = 0;

calibration_pts_ctrl = 0;
calibration_pts_ctrl_XYZ = true;

correction_dist_DLT = 1;
calcul_param_DLT = true;

% Affichage des figures
figure_1 = 1;                                                   % Points de controle
figure_4 = 0;                                                   % Profils laser 2D (non interp et interp)
figure_8 = true;                                                % interpolation de la géométrie
figure_16 = true; res_affichage = 2;                            % interpolation de la géométrie

%% EXECUTION
% boucle sur les dossiers images
for id_fold = 1:dossiers_a_traiter

    close all
    clearvars -except posiCam_stock T_stock ypr_stock PPuo_stock PPvo_stock Z_stock id_fold main_images_directory nameFolds...
        derniere_image pas_image ...
        calibration_pts_ctrl calibration_pts_ctrl_XYZ calibration_axes_lsr correction_dist_DLT ...
        calibration_pts_ctrl_sup_axes calibration_pts_ctrl_sup_axes_XYZ nb_interpolations ...
        interpolation_restit detection_des_zones_digue_et_axes detection_des_zones_eau ...
        decim cas_traitement_laser percentile_bin_img param_fond_canal calcul_param_DLT...
        std_prof_laser_max nb_med_image refraction coor_axes_inters_obj retrait_plan_laser...
        hauteurs_eau_rest temps_rest N_eau Seuil_refraction hauteur_eau_breche pas_img_med ...
        limite_px_retirer calibration_axes_lsr_XYZ support_laser decalage_a_la_base...
        retrait_refraction  marge_aval marge_amont puissance_coef_transition_ref...
        hauteurs_eau_rest_am hauteurs_eau_rest_av valeur_seuil_Cr_rouge...
        figure_1 figure_3 figure_4 figure_60 figure_7 figure_8 figure_16 res_affichage freq_affich

    i_clr_rouge = 0;
    Couleur_rouge = [];

    pathnames = strcat(main_images_directory, nameFolds{id_fold});
    display(pathnames)
    images0 = ls(pathnames);
    pathnames = strcat(pathnames,'\');
    addpath(pathnames);

    if derniere_image > 1
        images = images0(3:pas_image:derniere_image-2,:);
    else
        images = images0(3:pas_image:end-2,:);
    end

    nom_dossier = nameFolds{id_fold};

    %% Chargement des images
    fprintf('Chargement des images... \n')
    display(pathnames)

    nb_images = size(images,1);

    % importation des images
    image_rgb_full = imread(strcat(pathnames,images(1,:)));

    % calcul des dimensions de l'image
    X_px = size(image_rgb_full(:,:,1),2);
    Y_px = size(image_rgb_full(:,:,1),1);

    % centre de l'image
    u0 = X_px/2;
    v0 = Y_px/2;

    %% Determination manuelle des zones masques (digue, axe, eau_canal, eau_breche)
    %Determination_des_zones;

    %% Chargement des coordonnées monde des points de contrôle

    % figure des points de controle (ref. objet)
    if figure_1
        load('Points_control.mat','Points_control');
        figure(1)
        hold on
        box on
        grid on
        set(gca,'YDir','Reverse')
        view([-30 30])
        plot3(Points_control(:,2),Points_control(:,3),Points_control(:,4),':','Marker','o','MarkerSize',4)
        for i = 1:size(Points_control,1)
            text(Points_control(i,2),Points_control(i,3),Points_control(i,4),sprintf('  < %g',i))
            text(Points_control(i,2)+10,Points_control(i,3), ...
                Points_control(i,4)+7, ['(' num2str(Points_control(i,2)) ', ' ...
                num2str(Points_control(i,3)) ', ' num2str(Points_control(i,4)) ')'], 'FontSize', 6, 'Color', [0 .3 .7])
        end
        daspect([1 1 .75])
    end

    %% Localisation manuelle des points de controle
    if calibration_pts_ctrl
        load('Points_control.mat','Points_control');
        nb_pts = size(Points_control,1);
        Profilo_calibration_pts_ctrl;
    else
        % Chargement des coordonnées u, v (image) des points de controles
        load(sprintf('u_v_%s.mat',nameFolds{id_fold}),'u','v');
        % Chargement des coordonnées x, y, z (objet) des points de controles
        load('Points_control.mat','Points_control');
```

```
        nb_pts = size(Points_control,1);
    end

    % réécriture des coo. monde des points de contrôle
    x = Points_control(:,2)-1000;
    y = Points_control(:,3)-1000;
    z = Points_control(:,4)-100;

    %% Calculs des paramètres de la DLT
    if calcul_param_DLT
        fprintf('Calcul des paramètres de la DLT \n\n');

        Profilo_clcl_param_DLT;

        if figure_1
            plot3(x,y,z,':','Marker','.','MarkerSize',6,'Color',[.8 .8 .8])
            plot3(Camera(1),Camera(2),Camera(3),'sr','MarkerSize',15)
            plot3(Camera(1),Camera(2),Camera(3),'xk','MarkerSize',12)
            text(Camera(1)+17,Camera(2)+17,Camera(3)+17,'Camera')
            set(gca,'DataAspectRatio',[1 1 1])
            title('Volume de référence et caméra')
        end

        save(sprintf('Param_DLT_%s.mat',nameFolds{id_fold}),'L');

        [posiCam,T,ypr,PPuo,PPvo,Z]=DLTcameraPosition(L(1:11));

        posiCam_stock(:,id_fold) = posiCam(:,1);
        Z_stock(id_fold,1) = Z;
        PPuo_stock(id_fold,1) = PPuo;
        PPvo_stock(id_fold,1) = PPvo;
        ypr_stock(id_fold,:) = ypr(1,:);
        T_stock(((3*id_fold)-2):(3*id_fold),1:3) = T(1:3,1:3);

        save('posiCam.mat','posiCam_stock');
        save('Z.mat','Z_stock');
        save('PPuo.mat','PPuo_stock');
        save('PPvo.mat','PPvo_stock');
        save('ypr.mat','ypr_stock');
        save('T.mat','T_stock');

    else
        load('posiCam.mat','posiCam_stock');
        load('Z.mat','Z_stock');
        load('PPuo.mat','PPuo_stock');
        load('PPvo.mat','PPvo_stock');
        load('ypr.mat','ypr_stock');
        load('T.mat','T_stock');
    end
end

%% PARTIE 2 : RESTITUTION 3D DES POINTS DE VERIFICATION

fprintf('Restitution 3D \n\n');

% Importation des données
% ***********************

%Nombre de caméras
prompt = 'Combien de caméras sont utilisées? ';
nCam = input(prompt);

%Nombe de points homologues maximum
nHo=11;

%Paramètres L
L_param=zeros(16,1,nCam);

load Param_DLT_Canon.mat;
L_param(:,1,1) = L(:,1);
load Param_DLT_Lumix72.mat;
L_param(:,1,2) = L(:,1);
load Param_DLT_LumixGH4.mat;
L_param(:,1,3) = L(:,1);
%etc selon le nomnre de caméras

%Coordonnées du points principal pour chaque image
PP=zeros(1,2,nCam);

PP(1,1,1)=PPuo_stock(1,1);
PP(1,2,1)=PPvo_stock(1,1);
PP(1,1,2)=PPuo_stock(2,1);
PP(1,2,2)=PPvo_stock(2,1);
PP(1,1,3)=PPuo_stock(3,1);
PP(1,2,3)=PPvo_stock(3,1);
%etc selon le nomnre de caméras

%Coordonnées monde des points de vérification.
load GCPmonde.mat
GCPmonde(:,1)=GCPmonde(:,1)-1000;
GCPmonde(:,2)=GCPmonde(:,2)-1000;
GCPmonde(:,3)=GCPmonde(:,3)-100;

%Points homologues
homologue=zeros(nHo,2,nCam);

load hom_Canon.mat;
homologue(:,:,1)=hom(:,:);
load hom_Lumix72.mat;
homologue(:,:,2)=hom(:,:);
```

```
load hom_LumixGH4.mat;
homologue(:,:,3)=hom(:,:);
%etc selon le nomnre de caméras

% Restitution 3D
% **************

%Restitution des points de contrôle
DLT_Restitution_16p_modif;

% controle qualité

ErrorX=xyz(:,1)-GCPmonde(:,1);                              %résidus en x
ErrorY=xyz(:,2)-GCPmonde(:,2);                              %résidus en y
ErrorZ=xyz(:,3)-GCPmonde(:,3);                              %résidus en z
ErrorGlobal=(sum((ErrorX.^2+ErrorY.^2+ErrorZ.^2).^0.5))/nHo;  %RMSE

ErrorXYZ=(ErrorX.^2+ErrorY.^2+ErrorZ.^2).^0.5;  %distance d'erreur pour PV

%Plot

figure;

bbar16=bar(ErrorXYZ);
hold off
get(bbar16);
set(bbar16,'faceColor',[0 0.5 0.5],'EdgeColor','w','DisplayName', 'Figure - Résidus GCP L16');

xlabel('Points de vérfication');
ylabel('Résidus L16 (m)');
title('Résidus par GCP - L16 avec prise en compte la distorsion!!!');

hold off

% Exportations

%Exportation des points restitués

exportXYZ=xlswrite('XYZ',xyz);

%Exportation des rapports d'erreurs
exportErrorX=xlswrite('residusX',ErrorX);
exportErrorY=xlswrite('residusY',ErrorY);
exportErrorZ=xlswrite('residusZ',ErrorZ);

exportErrorXYZ=xlswrite('ErrorXYZ',ErrorXYZ);

exportErrorGlobal=xlswrite('RMSE',ErrorGlobal);

disp('Restitution effectuée')

%% PARTIE 3 : DETECTION DES POINTS HOMOLOGUES

fprintf('Détection des points homologues \n\n');

% SYNCHRONISATION
% ***************

erreur_totale_synch = (0.8 - 10/30) * 25;

fprintf('Intervalle de temps à traiter \n\n');
prompt = ('Borne inférieure en secondes: ');
temps_inf = input(prompt);
prompt = ('Borne supérieure en secondes: ');
temps_sup = input(prompt);
prompt = ('Nombre de pas (frames): ');
pas = input(prompt);

frames_inf_Lumix = temps_inf * 30;
frames_sup_Lumix = temps_sup * 30;

synch = 1;

for frames = frames_inf_Lumix:pas:frames_sup_Lumix

    synch_canon(synch,1) = round((fix(frames/30)*25)+(((1/30)*mod(frames,30))*25));
    frames_synch = synch_canon(synch,1) + ((synch_canon(synch,1) * erreur_totale_synch)/24321);
    synch_canon(synch,1) = round(frames_synch);
    synch = synch + 1;

end

taille_synch = size(synch_canon);

% TRAITEMENT
% **********

for id_fold = 1:nCam

    pas_image = 1;
    derniere_image = 0;                                          % zero pour aller jusqu'à la dernière image

    pathnames = strcat(main_images_directory, nameFolds{id_fold});
    display(pathnames)
    images0 = ls(pathnames);
    pathnames = strcat(pathnames,'\');
    addpath(pathnames);

    if derniere_image > 1
        images = images0(3:pas_image:derniere_image-2,:);
```

```
    else
        images = images0(3:pas_image:end-2,:);
    end

    nom_dossier = nameFolds{id_fold};

    display(pathnames)

    nb_images = size(images,1);

    if (id_fold == 1) %Centroïde de tous les flotteurs des images du Canon

        choix = 0;
        w = 1;

        for r = 1:taille_synch

            image_canon = synch_canon(r,1);

            image_rgb_full = imread(strcat(pathnames,images(image_canon,:)));

            classification_flotteur;

            nbrJauneCanon(w) = n_jaune;
            nbrOrangeCanon(w) = n_orange;

            Canon_data_jaune(w).id = image_canon;
            Canon_data_orange(w).id = image_canon;

            for i=1:n_jaune
                Canon_data_jaune(w).flotteur{i,1} =  data_grey_jaune.WeightedCentroid(i,1);
                Canon_data_jaune(w).flotteur{i,2} =  data_grey_jaune.WeightedCentroid(i,2);
            end

            for i=1:n_orange
                Canon_data_orange(w).flotteur{i,1} = data_grey_orange.WeightedCentroid(i,1);
                Canon_data_orange(w).flotteur{i,2} = data_grey_orange.WeightedCentroid(i,2);
            end
            w = w+1;
        end
    end

    if (id_fold == 2) %Droite épipolaire des images du Canon et LumixGH4 à partir des flotteurs du Lumix72

        choix = 0;
        w = 1;

        for m=frames_inf_Lumix:pas:frames_sup_Lumix

            image_rgb_full = imread(strcat(pathnames,images(m,:)));

            classification_flotteur;

            nbrJauneLumix72(w) = n_jaune;
            nbrOrangeLumix72(w) = n_orange;

            homologue_jaune(w).id = m;
            homologue_orange(w).id = m;

            data_epipo_jaune(w).id = m;
            data_epipo_orange(w).id = m;

            for o=1:n_jaune

                x_centroid = data_grey_jaune.WeightedCentroid(o,1);
                y_centroid = data_grey_jaune.WeightedCentroid(o,2);

                homologue_jaune(w).lumix72{o,1} = x_centroid;
                homologue_jaune(w).lumix72{o,2} = y_centroid;

                camera1 = 2;
                camera2 = 1;
                geometrie_epipolaire;

                data_epipo_jaune(w).pente{o,1} = pente;
                data_epipo_jaune(w).intercept{o,1} = intercept;

                camera1 = 2;
                camera2 = 3;
                geometrie_epipolaire;

                data_epipo_jaune(w).pente{o,2} = pente;
                data_epipo_jaune(w).intercept{o,2} = intercept;

            end

            for l=1:n_orange

                x_centroid = data_grey_orange.WeightedCentroid(l,1);
                y_centroid = data_grey_orange.WeightedCentroid(l,2);

                homologue_orange(w).lumix72{l,1} = x_centroid;
                homologue_orange(w).lumix72{l,2} = y_centroid;

                camera1 = 2;
                camera2 = 1;
                geometrie_epipolaire;

                data_epipo_orange(w).pente{l,1} = pente;
                data_epipo_orange(w).intercept{l,1} = intercept;
```

```
451
452 -                 camera1 = 2;
453 -                 camera2 = 3;
454 -                 geometrie_epipolaire;
455
456 -                 data_epipo_orange(w).pente{1,2} = pente;
457 -                 data_epipo_orange(w).intercept{1,2} = intercept;
458 -             end
459 -             w = w + 1;
460 -         end
461 -     end
462
463 -     echantillonage_jaune = 1;
464 -     echantillonage_orange = 1;
465
466 -     if (id_fold == 3) %Candidats dans les images du LumixGH4
467
468 -         choix = 0;
469 -         w = 1;
470
471 -         for e = frames_inf_Lumix:pas:frames_sup_Lumix
472
473 -             image_rgb_full = imread(strcat(pathnames,images(e,:)));
474
475 -             classification_flotteur;
476
477 -             candidat_jaune(w).id = e;
478 -             candidat_orange(w).id = e;
479
480 -             nHo = nbrJauneLumix72(w);
481 -             homologue = zeros(nHo,2,nCam);
482
483 -             %Flotteurs jaunes
484 -             %***************
485
486 -             for h = 1:nbrJauneLumix72(w) %pour une droite d'une image du LumixGH4
487
488 -                 t = 1;
489
490 -                 for f = 1:n_jaune %candidats possibles
491
492 -                     erreur_coplaneite_jaune = abs((Y_px - data_grey_jaune.WeightedCentroid(f,2)) - ...
493 -                         data_epipo_jaune(w).pente{h,2}*data_grey_jaune.WeightedCentroid(f,1) - ...
494 -                         data_epipo_jaune(w).intercept{h,2})/(sqrt(1+data_epipo_jaune(w).pente{h,2}^2));
495
496 -                     if(round(erreur_coplaneite_jaune) <= 20)
497 -                         candidat_jaune(w).flotteur_x{h,t} = data_grey_jaune.WeightedCentroid(f,1);
498 -                         candidat_jaune(w).flotteur_y{h,t} = data_grey_jaune.WeightedCentroid(f,2);
499 -                         candidat_jaune(w).nombre{h,1} = t;
500 -                         t = t+1;
501 -                     end
502 -                 end
503
504 -                 %les droites épipolaires des images du Canon à partir des
505 -                 %candidats identifiés ci-dessus
506
507 -                 camera1 = 3;
508 -                 camera2 = 1;
509
510 -                 nbr_candidat_jaune = t-1;
511
512 -                 if (nbr_candidat_jaune ~= 0)
513
514 -                     clear distance_jaune;
515 -                     for l = 1:nbr_candidat_jaune% Droite épipo image du Canon
516
517 -                         x_centroid = candidat_jaune(w).flotteur_x{h,l};
518 -                         y_centroid = candidat_jaune(w).flotteur_y{h,l};
519 -                         geometrie_epipolaire;
520
521 -                         syms x y
522 -                         epipo1 = data_epipo_jaune(w).pente{h,l}*x + data_epipo_jaune(w).intercept{h,l} == y;
523 -                         epipo2 = pente*x + intercept == y;
524 -                         resolut = solve(epipo1,epipo2);
525
526 -                         candidat_jaune(w).intersection_x{h,l} = double(resolut.x);
527 -                         candidat_jaune(w).intersection_y{h,l} = Y_px - double(resolut.y);
528
529 -                         for i=1:nbrJauneCanon(w)
530 -                             distance_jaune(i,l) = sqrt(((Canon_data_jaune(w).flotteur{i,1} - ...
531                                 candidat_jaune(w).intersection_x{h,l})^2)+((Canon_data_jaune(w).flotteur{i,2}...
532                                 - candidat_jaune(w).intersection_y{h,l})^2));
533 -                         end
534
535 -                         [min1,indice1] = min(distance_jaune(:,l));
536
537 -                         candidat_jaune(w).proche_inter_min{h,l} = min1;
538 -                         candidat_jaune(w).proche_inter_indice{h,l} = indice1;
539 -                     end
540
541 -                     camera1 = 2;
542 -                     camera2 = 1;
543 -                     erreur_coplanarite_jaune;
544 -                     [min_produit,indice_produit] = min(produit_scalaire);
545
546 -                     %Point homologues
547 -                     bon_candidat = candidat_jaune(w).proche_inter_indice{h,indice_produit};
548
549 -                     homologue_jaune(w).canon{h,1} = Canon_data_jaune(w).flotteur{bon_candidat,1};
550 -                     homologue_jaune(w).canon{h,2} = Canon_data_jaune(w).flotteur{bon_candidat,2};
```

```
                    homologue(h,1,1) = homologue_jaune(w).canon(h,1);
                    homologue(h,2,1) = homologue_jaune(w).canon(h,2);

                    homologue_jaune(w).lumixGH4(h,1) = candidat_jaune(w).flotteur_x(h,indice_produit);
                    homologue_jaune(w).lumixGH4(h,2) = candidat_jaune(w).flotteur_y(h,indice_produit);
                    homologue(h,1,3) = homologue_jaune(w).lumixGH4(h,1);
                    homologue(h,2,3) = homologue_jaune(w).lumixGH4(h,2);

                    homologue(h,1,2) = homologue_jaune(w).lumix72(h,1);
                    homologue(h,2,2) = homologue_jaune(w).lumix72(h,2);
                end
            end

            %Restitution 3D
            DLT_Restitution_16p_modif;

            jaune_xyz(w).id = w;

            for i = 1:nHo
                jaune_xyz(w).position(i,1) = xyz(i,1);
                jaune_xyz(w).position(i,2) = xyz(i,2);
                jaune_xyz(w).position(i,3) = xyz(i,3) - 0.011176242;
                jaune_xyz(w).position(i,4) = 0;
                if (jaune_xyz(w).position(i,3) < 1.5)
                    hauteur_jaune_x(echantillonage_jaune,1) = xyz(i,1);
                    hauteur_jaune_y(echantillonage_jaune,1) = xyz(i,2);
                    hauteur_jaune_z(echantillonage_jaune,1) = xyz(i,3) - 0.00067;
                    echantillonage_jaune = echantillonage_jaune + 1;
                end
            end

            nHo = nbrOrangeLumix72(w);
            homologue = zeros(nHo,2,nCam);

            %Flotteurs oranges
            %*****************

            for h = 1:nbrOrangeLumix72(w) %pour une droite d'une image du LumixGH4

                t = 1;

                for f = 1:n_orange %candidats possibles

                    erreur_coplaneite_orange = abs((Y_px - data_grey_orange.WeightedCentroid(f,2)) - ...
                        data_epipo_orange(w).pente(h,2)*data_grey_orange.WeightedCentroid(f,1) - ...
                        data_epipo_orange(w).intercept(h,2))/(sqrt(1+data_epipo_orange(w).pente(h,2)^2));

                    if(round(erreur_coplaneite_orange) <= 20)
                        candidat_orange(w).flotteur_x(h,t) = data_grey_orange.WeightedCentroid(f,1);
                        candidat_orange(w).flotteur_y(h,t) = data_grey_orange.WeightedCentroid(f,2);
                        candidat_orange(w).nombre(h,1) = t;
                        t = t+1;
                    end
                end

                %les droites épipolaires des images du Canon à partir des
                %candidats identifiés ci-dessus

                camera1 = 3;
                camera2 = 1;

                nbr_candidat_orange = t-1;

                if (nbr_candidat_orange ~= 0)

                    clear distance_orange;
                    for l = 1:nbr_candidat_orange%Droite épipo image du Canon

                        x_centroid = candidat_orange(w).flotteur_x(h,l);
                        y_centroid = candidat_orange(w).flotteur_y(h,l);
                        geometrie_epipolaire;

                        syms x y
                        epipo1 = data_epipo_orange(w).pente(h,1)*x + data_epipo_orange(w).intercept(h,1) == y;
                        epipo2 = pente*x + intercept == y;
                        resolut = solve(epipo1,epipo2);

                        candidat_orange(w).intersection_x(h,l) = double(resolut.x);
                        candidat_orange(w).intersection_y(h,l) = Y_px - double(resolut.y);

                        for i=1:nbrOrangeCanon(w)
                            distance_orange(i,l) = sqrt(((Canon_data_orange(w).flotteur(i,1} - ...
                                candidat_orange(w).intersection_x(h,l})^2)+((Canon_data_orange(w).flotteur(i,2} - ...
                                candidat_orange(w).intersection_y(h,l})^2));
                        end

                        [min1,indice1] = min(distance_orange(:,l));

                        candidat_orange(w).proche_inter_min(h,l} = min1;
                        candidat_orange(w).proche_inter_indice(h,l} = indice1;
                    end

                    camera1 = 2;
                    camera2 = 1;
                    erreur_coplanarite_orange;
                    [min_produit,indice_produit] = min(produit_scalaire);

                    %Point homologues
                    bon_candidat = candidat_orange(w).proche_inter_indice(h,indice_produit};
                    homologue_orange(w).canon(h,1} = Canon_data_orange(w).flotteur(bon_candidat,1};
```

```
                    homologue_orange(w).canon{h,2} = Canon_data_orange(w).flotteur{bon_candidat,2};
                    homologue(h,1,1) = homologue_orange(w).canon{h,1};
                    homologue(h,2,1) = homologue_orange(w).canon{h,2};

                    homologue_orange(w).lumixGH4{h,1} = candidat_orange(w).flotteur_x{h,indice_produit};
                    homologue_orange(w).lumixGH4{h,2} = candidat_orange(w).flotteur_y{h,indice_produit};
                    homologue(h,1,3) = homologue_orange(w).lumixGH4{h,1};
                    homologue(h,2,3) = homologue_orange(w).lumixGH4{h,2};

                    homologue(h,1,2) = homologue_orange(w).lumix72{h,1};
                    homologue(h,2,2) = homologue_orange(w).lumix72{h,2};
                end
            end

            %Restitution 3D
            DLT_Restitution_16p_modif;

            orange_xyz(w).id = w;

            for i = 1:nHo
                orange_xyz(w).position{i,1} = xyz(i,1);
                orange_xyz(w).position{i,2} = xyz(i,2);
                orange_xyz(w).position{i,3} = xyz(i,3) - 0.011176242;
                orange_xyz(w).position{i,4} = 0;
                if (orange_xyz(w).position{i,3} < 1.5)
                    hauteur_orange_x(echantillonage_orange,1) = xyz(i,1);
                    hauteur_orange_y(echantillonage_orange,1) = xyz(i,2);
                    hauteur_orange_z(echantillonage_orange,1) = xyz(i,3) - 0.00067;
                    echantillonage_orange = echantillonage_orange + 1;
                end
            end

            w = w +1;
        end
    end
end

% REPRESENTATION GRAPHIQUE STATIQUE HAUTEUR D EAU
% -----------------------------------------------

hauteur_eau_x(1:(echantillonage_jaune-1),1) = hauteur_jaune_x(:,1);
hauteur_eau_x((echantillonage_jaune:((echantillonage_jaune-1)+(echantillonage_orange-1))),1) = hauteur_orange_x(:,1);

hauteur_eau_y(1:(echantillonage_jaune-1),1) = hauteur_jaune_y(:,1);
hauteur_eau_y((echantillonage_jaune:((echantillonage_jaune-1)+(echantillonage_orange-1))),1) = hauteur_orange_y(:,1);

hauteur_eau_z(1:(echantillonage_jaune-1),1) = hauteur_jaune_z(:,1);
hauteur_eau_z((echantillonage_jaune:((echantillonage_jaune-1)+(echantillonage_orange-1))),1) = hauteur_orange_z(:,1);

%Restitution 3D des flotteurs
figure
scatter3(hauteur_eau_x,hauteur_eau_y,hauteur_eau_z,'MarkerEdgeColor','k','MarkerFaceColor',[0 .75 .75])
view(-30,10)

%Surface de tendance
figure
surface_tendance = fit([hauteur_eau_x,hauteur_eau_y],hauteur_eau_z, 'poly44','Robust','on')
plot(surface_tendance, [hauteur_eau_x,hauteur_eau_y],hauteur_eau_z)
title('Surface de tendance');
zlabel('Altitude (m)');

figure
plot(surface_tendance,[hauteur_eau_x,hauteur_eau_y],hauteur_eau_z,'Style','Residuals')
z_estime = surface_tendance(hauteur_eau_x,hauteur_eau_y)
z_residu = hauteur_eau_z - z_estime;
title('Résidus par rapport à la surface de tendance');
zlabel('Altitude (m)');


%% Partie 4 : Calcul des trajectoires 3D
%---------------------------------------

%TRAITEMENT
%----------

%FLOTTEURS JAUNES

t_jaune = 1;

for e = 1:(taille_synch-1) %1:n-1

    nbr_position_image0 = size(jaune_xyz(e).position);

    for i = 1:nbr_position_image0(1,1)

        if(jaune_xyz(e).position{i,4} ~= -1)

            trajectoire_jaune(t_jaune).trajectoire{1,1} = jaune_xyz(e).position{i,1};
            trajectoire_jaune(t_jaune).trajectoire{1,2} = jaune_xyz(e).position{i,2};
            trajectoire_jaune(t_jaune).trajectoire{1,3} = jaune_xyz(e).position{i,3};
            jaune_xyz(e).position{i,4} = -1;

            o = 2;

            previous = e;

            for j = (e+1):taille_synch %e+1:nbr_images

                nbr_position_image1 = size(jaune_xyz(j).position);

```

```
                clear distance_flotteur_jaune;

                for k = 1:nbr_position_image1(1,1)
                    distance_flotteur_jaune(k,1) = sqrt(((jaune_xyz(e).position{i,1} - jaune_xyz(j).position{k,1})^2) + ...
                        ((jaune_xyz(e).position{i,2} - jaune_xyz(j).position{k,2})^2) + ...
                        ((jaune_xyz(e).position{i,3} - jaune_xyz(j).position{k,3})^2));
                end

                [min_distance_jaune,indice_distance_jaune] = min(distance_flotteur_jaune(:,1));

                if ((min_distance_jaune < 0.14) && (j == previous+1))

                trajectoire_jaune(t_jaune).trajectoire{o,1} = jaune_xyz(j).position{indice_distance_jaune,1};
                trajectoire_jaune(t_jaune).trajectoire{o,2} = jaune_xyz(j).position{indice_distance_jaune,2};
                trajectoire_jaune(t_jaune).trajectoire{o,3} = jaune_xyz(j).position{indice_distance_jaune,3};
                jaune_xyz(j).position{indice_distance_jaune,4} = -1;
                previous = j;

                end
                o = o+1;
            end
            t_jaune = t_jaune+1;
        end
    end
end


% FLOTTEURS ORANGES

t_orange = 1;

for e = 1:(taille_synch-1)%1:n-1

    nbr_position_i = size(orange_xyz(e).position);

    for i = 1:nbr_position_i(1,1)

        if(orange_xyz(e).position{i,4} ~= -1)

            trajectoire_orange(t_orange).trajectoire{1,1} = orange_xyz(e).position{i,1};
            trajectoire_orange(t_orange).trajectoire{1,2} = orange_xyz(e).position{i,2};
            trajectoire_orange(t_orange).trajectoire{1,3} = orange_xyz(e).position{i,3};
            orange_xyz(e).position{i,4} = -1;

            o = 2;

            previous = e;

            for j = (e+1):taille_synch% e+1:nbr_images

                nbr_position_j = size(orange_xyz(j).position);

                clear distance_flotteur_orange;

                for k = 1:nbr_position_j(1,1)
                    distance_flotteur_orange(k,1) = sqrt(((orange_xyz(e).position{i,1} - orange_xyz(j).position{k,1})^2) ...
                        + ((orange_xyz(e).position{i,2} - orange_xyz(j).position{k,2})^2) + ...
                        ((orange_xyz(e).position{i,3} - orange_xyz(j).position{k,3})^2));
                end

                [min_distance_orange,indice_distance_orange] = min(distance_flotteur_orange(:,1));


                if ((min_distance_orange < 0.14) && (j == previous+1))

                    trajectoire_orange(t_orange).trajectoire{o,1} = orange_xyz(j).position{indice_distance_orange,1};
                    trajectoire_orange(t_orange).trajectoire{o,2} = orange_xyz(j).position{indice_distance_orange,2};
                    trajectoire_orange(t_orange).trajectoire{o,3} = orange_xyz(j).position{indice_distance_orange,3};
                    orange_xyz(j).position{indice_distance_orange,4} = -1;
                    previous = j;

                end
                o = o+1;
            end
            t_orange = t_orange+1;
        end
    end
end

%REPRESENTATION GRAPHIQUE TRAJECTOIRE
%-------------------------------------

figure

for i=1:(t_jaune-1)

    clear test;
    test = (trajectoire_jaune(i).trajectoire);
    test = cell2mat(test);

    test3 = (test(:,3));
    test1 = (test(:,1));
    test2 = (test(:,2));

    hold on
    fnplt(cscvn(test(:,[1:end 1])'),'y',2)
end

for i=1:(t_orange-1)
```

```
    clear test;
    test = trajectoire_orange(i).trajectoire;
    test = cell2mat(test);

    test3 = (test(:,3));
    test1 = (test(:,1));
    test2 = (test(:,2));

    fnplt(cscvn(test(:,[1:end 1])'),'r',2)
end


%% Partie 5 : Calcul des champs de vitesses
%-----------------------------------------

%TRAITEMENTS
%-----------

echantillon_vitesse = 1;

%FLOTTEURS JAUNES

for i=1:(t_jaune-1)

    nbr_position_trajectoire = size(trajectoire_jaune(i).trajectoire);
    nbr_position_trajectoire = nbr_position_trajectoire(1,1);

    if (nbr_position_trajectoire ~= 1)

        for j=2:nbr_position_trajectoire

            position_flotteurs(echantillon_vitesse,1) = (trajectoire_jaune(i).trajectoire{j-1,1} + trajectoire_jaune(i).trajectoire{j,1})/2;
            position_flotteurs(echantillon_vitesse,2) = (trajectoire_jaune(i).trajectoire{j-1,2} + trajectoire_jaune(i).trajectoire{j,2})/2;
            position_flotteurs(echantillon_vitesse,3) = (trajectoire_jaune(i).trajectoire{j-1,3} + trajectoire_jaune(i).trajectoire{j,3})/2;

            vitesse_jaune(i).variation_position{j-1,1} = trajectoire_jaune(i).trajectoire{j,1} - trajectoire_jaune(i).trajectoire{j-1,1};
            vitesse_jaune(i).variation_position{j-1,2} = trajectoire_jaune(i).trajectoire{j,2} - trajectoire_jaune(i).trajectoire{j-1,2};
            vitesse_jaune(i).variation_position{j-1,3} = trajectoire_jaune(i).trajectoire{j,3} - trajectoire_jaune(i).trajectoire{j-1,3};

            vitesse_jaune(i).vitesse{j-1,1} = vitesse_jaune(i).variation_position{j-1,1}/(pas/30);
            vitesse_jaune(i).vitesse{j-1,2} = vitesse_jaune(i).variation_position{j-1,2}/(pas/30);
            vitesse_jaune(i).vitesse{j-1,3} = vitesse_jaune(i).variation_position{j-1,3}/(pas/30);

            position_flotteurs(echantillon_vitesse,4) = sqrt((vitesse_jaune(i).vitesse{j-1,1})^2 + (vitesse_jaune(i).vitesse{j-1,2})^2 +...
                (vitesse_jaune(i).vitesse{j-1,3})^2);

            echantillon_vitesse = echantillon_vitesse + 1;
        end
    end
end

%FLOTTEURS ORANGES

for i=1:(t_orange-1)

    nbr_position_trajectoire = size(trajectoire_orange(i).trajectoire);
    nbr_position_trajectoire = nbr_position_trajectoire(1,1);

    if (nbr_position_trajectoire ~= 1)

        for j=2:nbr_position_trajectoire

            position_flotteurs(echantillon_vitesse,1) = (trajectoire_orange(i).trajectoire{j-1,1} + trajectoire_orange(i).trajectoire{j,1})/2;
            position_flotteurs(echantillon_vitesse,2) = (trajectoire_orange(i).trajectoire{j-1,2} + trajectoire_orange(i).trajectoire{j,2})/2;
            position_flotteurs(echantillon_vitesse,3) = (trajectoire_orange(i).trajectoire{j-1,3} + trajectoire_orange(i).trajectoire{j,3})/2;

            vitesse_orange(i).variation_position{j-1,1} = trajectoire_orange(i).trajectoire{j,1} - trajectoire_orange(i).trajectoire{j-1,1};
            vitesse_orange(i).variation_position{j-1,2} = trajectoire_orange(i).trajectoire{j,2} - trajectoire_orange(i).trajectoire{j-1,2};
            vitesse_orange(i).variation_position{j-1,3} = trajectoire_orange(i).trajectoire{j,3} - trajectoire_orange(i).trajectoire{j-1,3};

            vitesse_orange(i).vitesse{j-1,1} = vitesse_orange(i).variation_position{j-1,1}/(pas/30);
            vitesse_orange(i).vitesse{j-1,2} = vitesse_orange(i).variation_position{j-1,2}/(pas/30);
            vitesse_orange(i).vitesse{j-1,3} = vitesse_orange(i).variation_position{j-1,3}/(pas/30);

            position_flotteurs(echantillon_vitesse,4) = sqrt((vitesse_orange(i).vitesse{j-1,1})^2 + (vitesse_orange(i).vitesse{j-1,2})^2 + ...
                (vitesse_orange(i).vitesse{j-1,3})^2);

            echantillon_vitesse = echantillon_vitesse + 1;
        end
    end
end

%REPRESENTATION GRAPHIQUE
%------------------------

%Paramètre L de calibration de la troisième caméra
camera1 = 1;
camera2 = 3;
geometrie_epipolaire;

%Coord. images de la position des flotteurs
for i=1:echantillon_vitesse-1
    coord_image(i,1) = ((L(1,1)*position_flotteurs(i,1)) + (L(2,1)*position_flotteurs(i,2)) + (L(3,1)*position_flotteurs(i,3)) + ...
        L(4,1))/((L(9,1)*position_flotteurs(i,1)) + (L(10,1)*position_flotteurs(i,2)) + (L(11,1)*position_flotteurs(i,3))+1);
    coord_image(i,2) = ((L(5,1)*position_flotteurs(i,1)) + (L(6,1)*position_flotteurs(i,2)) + (L(7,1)*position_flotteurs(i,3)) + ...
        L(8,1))/((L(9,1)*position_flotteurs(i,1)) + (L(10,1)*position_flotteurs(i,2)) + (L(11,1)*position_flotteurs(i,3))+1);
    coord_image(i,2) = Y_px - coord_image(i,2);
end

%Statistiques
```

```
    figure

    boxplot(position_flotteurs(:,end));
    quantile_25 = quantile(position_flotteurs(:,4),0.25);
    quantile_75 = quantile(position_flotteurs(:,4),0.75);
    ecart = quantile_75 - quantile_25;

    vitesse_max = quantile_75 + 1.5*ecart;

    %Prendre la dernière image traitée dans l'intervalle de temps
    figure

    image_fond = imread('C:\Users\userr\Desktop\ULg\2 Master\Mémoire\Code Manon\Images\LumixGH4\P2420005 21686.jpg');%A changer par l'utilisateur
    imagesc(image_fond);

    hold on

    for i = 1:echantillon_vitesse-1
        if position_flotteurs(i,4)<=vitesse_max
        grid on
        box on
        daspect([1 1 1])
        scatter3(coord_image(i,1),coord_image(i,2),0,25,position_flotteurs(i,4),'o','Filled')
        colormap jet
        cb = colorbar;
        cb.Label.String = 'Vitesse (m/s)';
        end
    end

    axis off
    view(-30,10)
```