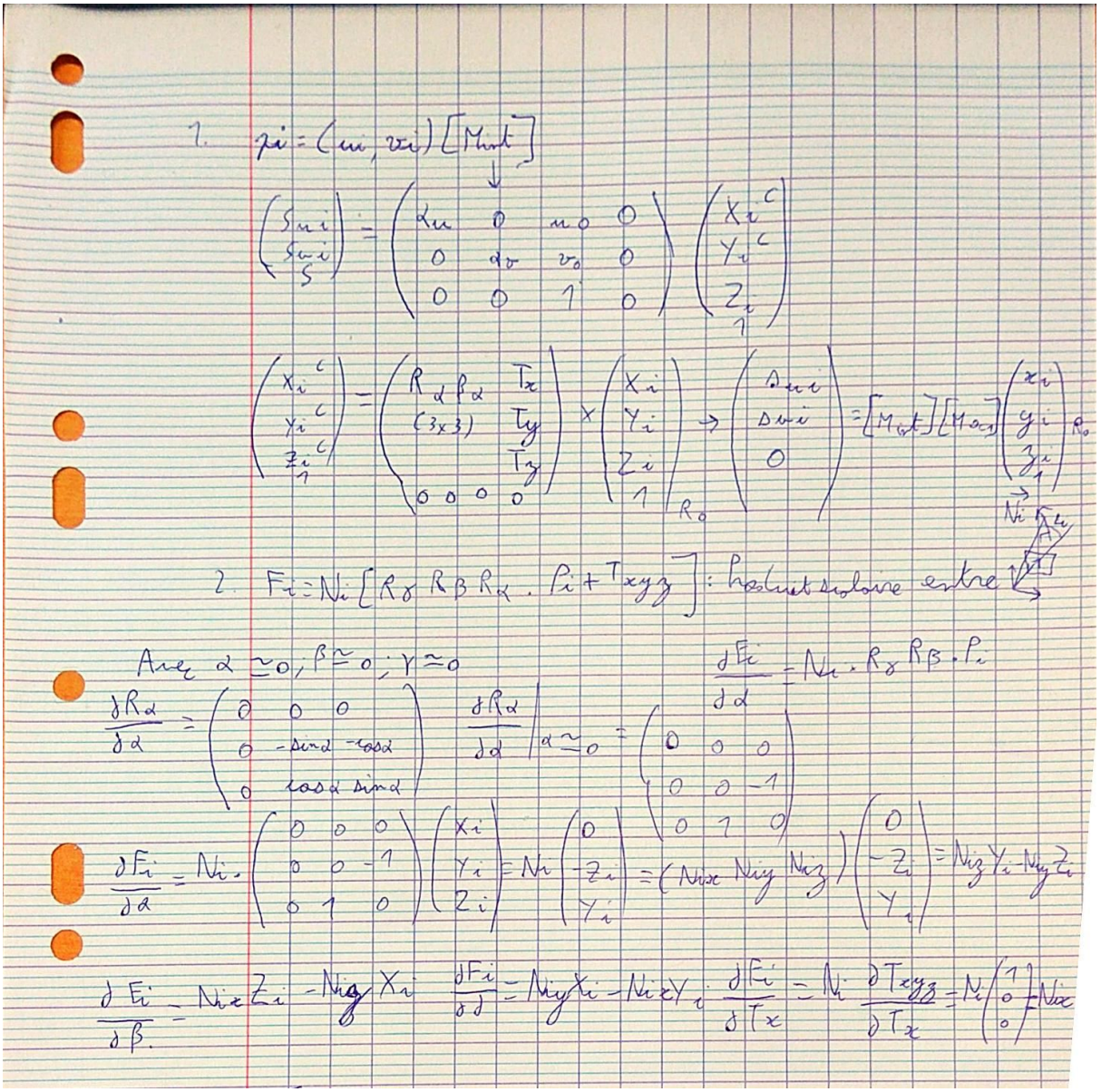
Rapport TP

TP Vision 3D - Localisation monoculaire

S9 -2017

Yoann Fleytoux
Aurélien Bernier

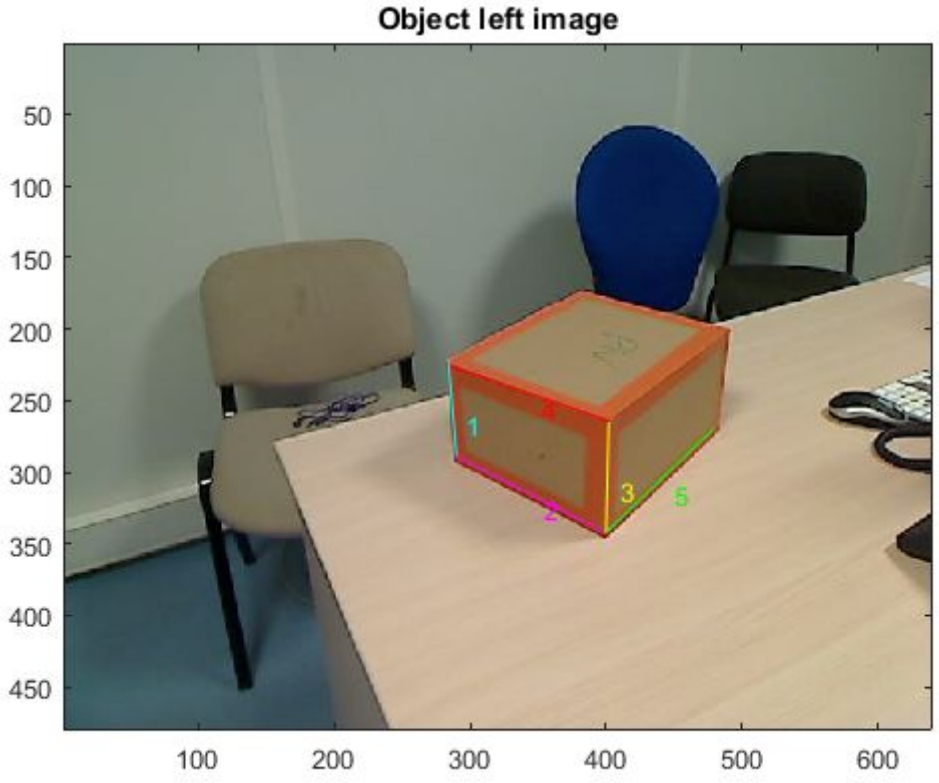
Formalisation sous-jacente



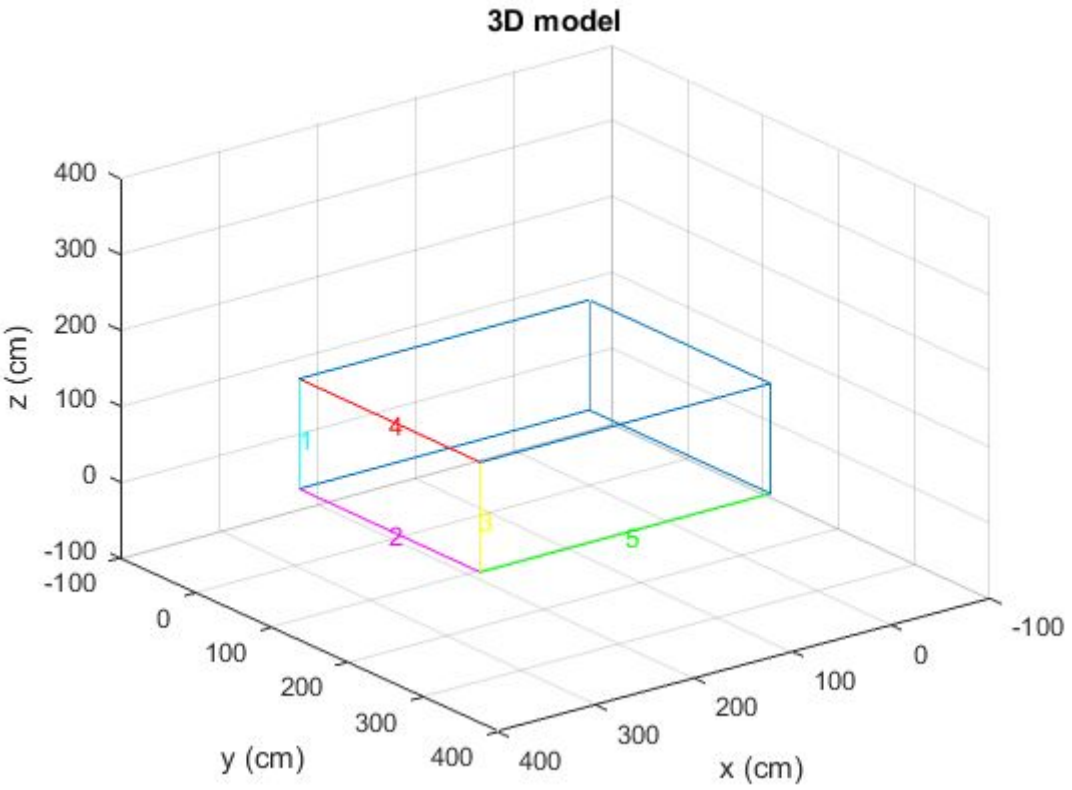
Implémentation MATLAB

Dans ce tp, le but était de faire correspondre un modèle CAO à sa position et orientation dans une image 2D fournie.

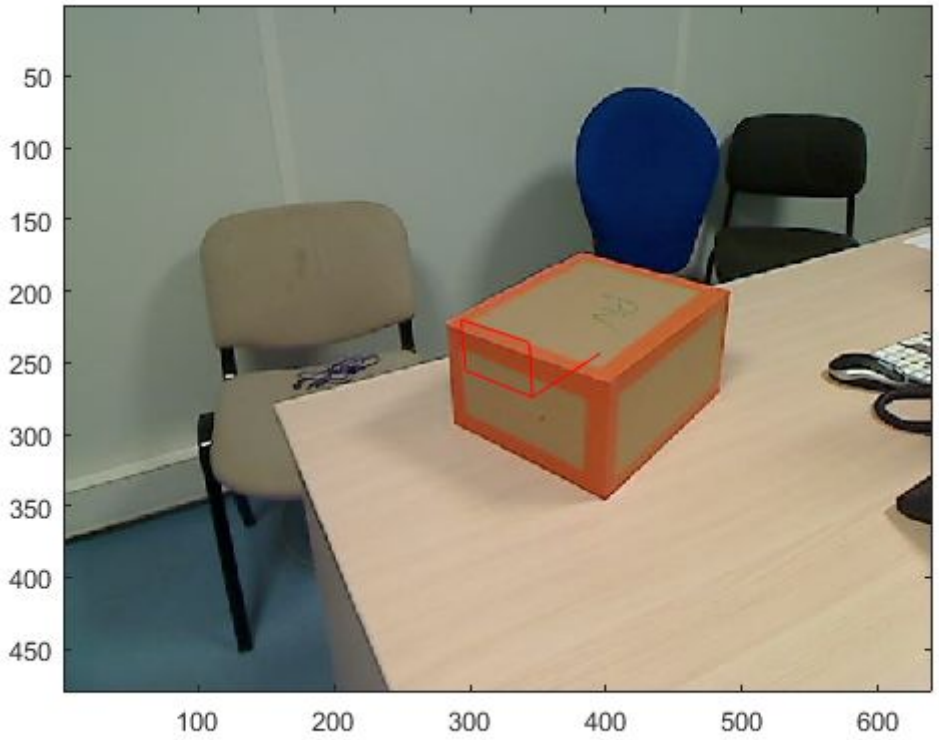
L'utilisateur sélectionne les segments du modèle sur l'image 2D.



L'utilisateur sélectionne les segments dans le même ordre sur le modèle CAO.



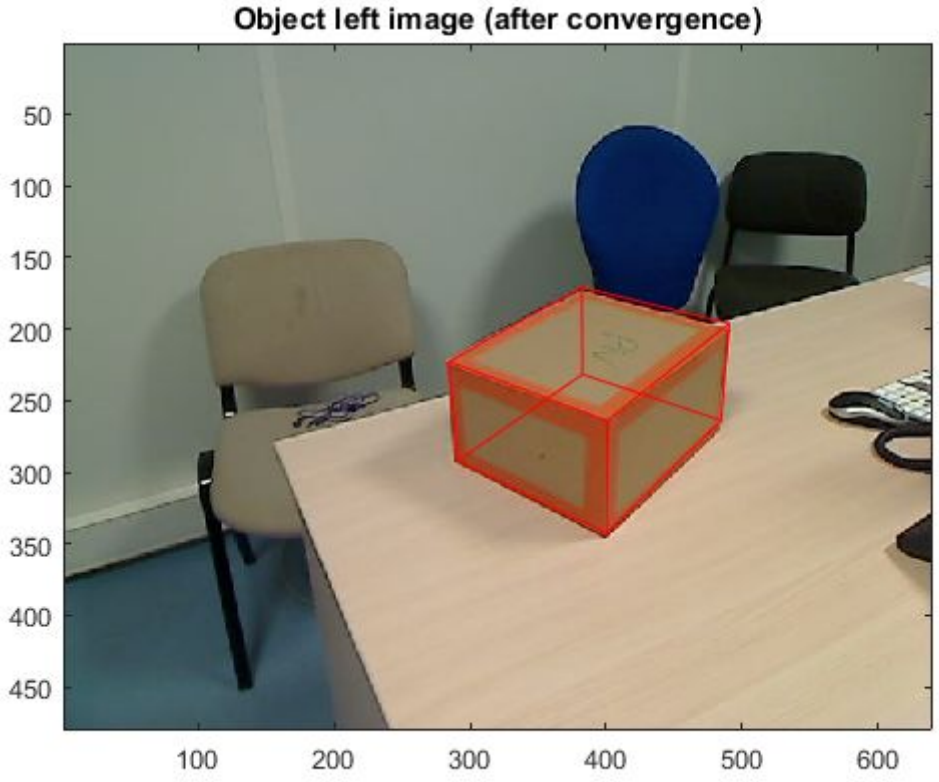
Les paramètres intrinsèques que nous avons ne sont pas assez bons pour une bonne correspondance. On va donc chercher à les optimiser :



Dans un premier temps on calcule les normales aux plans formés par les segments sélectionnés par l'utilisateur et la caméra.

On va ensuite former une boucle qui se terminera quand les 6 paramètres intrinsèques auront étés assez optimisés (en fonction d'un seuil sur la valeur renvoyée par la fonction d'évaluation du critère). A chaque boucle, on utilisera une matrice Jacobienne (qui permet de montrer l'impact de l'évolution des 6 paramètres de la matrice intrinsèque sur notre critère).

Le critère évalué est le suivant : les normales calculées précédemment sont multipliées par les segments du modèle. Si la normale et le segment appartiennent au même plan, le critère est 0.



Nous sommes allés jusqu'à la fin de la question 5.

Code MATLAB

```
%-------- MAIN PROGRAM -------------------------------------
function main

close all; clear all;
% Global variable
global App3d_ini;  % App3d=[ X(1) X(2) Y(1) Y(2) Z(1) Z(2) ] initial
coordinates
global App3d_mod;  % App3d=[ X(1) X(2) Y(1) Y(2) Z(1) Z(2) ] modified
coordinates
global App2d;      % App2d=[ normale.x normale.y normale.z ] normal of the
interpretation plane

% Matching number
EPS=0.5; N=5;      % N: Number of edges to be manually chosen

% 3D Model definition
L=240; W=295; H=145;
modP=[W W 0 W W 0 0 0; 0 L L 0 L L 0 0; 0 0 0 H H H H 0; 1 1 1 1 1 1 1 1];
modA=[modP(1,1) modP(2,1) modP(3,1) modP(1,2) modP(2,2) modP(3,2);
      modP(1,2) modP(2,2) modP(3,2) modP(1,3) modP(2,3) modP(3,3);
      modP(1,1) modP(2,1) modP(3,1) modP(1,4) modP(2,4) modP(3,4);
      modP(1,2) modP(2,2) modP(3,2) modP(1,5) modP(2,5) modP(3,5);
      modP(1,3) modP(2,3) modP(3,3) modP(1,6) modP(2,6) modP(3,6);
      modP(1,4) modP(2,4) modP(3,4) modP(1,7) modP(2,7) modP(3,7);
      modP(1,6) modP(2,6) modP(3,6) modP(1,7) modP(2,7) modP(3,7);
      modP(1,5) modP(2,5) modP(3,5) modP(1,6) modP(2,6) modP(3,6);
      modP(1,5) modP(2,5) modP(3,5) modP(1,4) modP(2,4) modP(3,4);
      modP(1,8) modP(2,8) modP(3,8) modP(1,1) modP(2,1) modP(3,1);
      modP(1,8) modP(2,8) modP(3,8) modP(1,7) modP(2,7) modP(3,7);
      modP(1,8) modP(2,8) modP(3,8) modP(1,3) modP(2,3) modP(3,3)];

% Plot the model (the box)
figure(1);
plot3(0, 0, 0);
ll1                                                                    =
line([modA(1,1),modA(1,4)],[modA(1,2),modA(1,5)],[modA(1,3),modA(1,6)]);
ll2                                                                    =
line([modA(2,1),modA(2,4)],[modA(2,2),modA(2,5)],[modA(2,3),modA(2,6)]);
ll3                                                                    =
line([modA(3,1),modA(3,4)],[modA(3,2),modA(3,5)],[modA(3,3),modA(3,6)]);
ll4                                                                    =
line([modA(4,1),modA(4,4)],[modA(4,2),modA(4,5)],[modA(4,3),modA(4,6)]);
ll5                                                                    =
line([modA(5,1),modA(5,4)],[modA(5,2),modA(5,5)],[modA(5,3),modA(5,6)]);
ll6                                                                    =
line([modA(6,1),modA(6,4)],[modA(6,2),modA(6,5)],[modA(6,3),modA(6,6)]);
ll7                                                                    =
line([modA(7,1),modA(7,4)],[modA(7,2),modA(7,5)],[modA(7,3),modA(7,6)]);
```

```
ll8                                                                    =
line([modA(8,1),modA(8,4)],[modA(8,2),modA(8,5)],[modA(8,3),modA(8,6)]);
ll9                                                                    =
line([modA(9,1),modA(9,4)],[modA(9,2),modA(9,5)],[modA(9,3),modA(9,6)]);
ll10                                                                   =
line([modA(10,1),modA(10,4)],[modA(10,2),modA(10,5)],[modA(10,3),modA(10,6)])
;
ll11                                                                   =
line([modA(11,1),modA(11,4)],[modA(11,2),modA(11,5)],[modA(11,3),modA(11,6)])
;
ll12                                                                   =
line([modA(12,1),modA(12,4)],[modA(12,2),modA(12,5)],[modA(12,3),modA(12,6)])
;

% Reverse x and y axis labels to position the inner most vertice at (0, 0, 0)
set(gca,'XDir','reverse');
set(gca,'YDir','reverse');
axis([-10 40 -10 40 -10 40]*10);
grid on;
xlabel('x (cm)'); ylabel('y (cm)'); zlabel('z (cm)'); title '3D model';

% Calibration left camera
%param intrinsèque alpha, betha, gamma, (angle euler), Tu,Tv,Tw
u0_1=310.740; v0_1=217.256; alphaU_1=529.654; alphaV_1=527.635; %camera1
% Left image loading
image_l=imread('imgLeft.png', 'png');
figure(2); colormap(gray(256));
image(image_l); title 'Object left image';
hold on;

% Calibration right camera
u0_2=310.731; v0_2=214.007; alphaU_2=531.037; alphaV_2=528.861;%camera2
% Right image loading
image_r=imread('imgRight.png', 'png');

% Stereo rig
matPos_cl_cr=[0.9999   -0.0076   0.0047   -95.3538;0.0075   0.9999   -0.0034
-2.2704;-0.0046 0.0035 0.9999 -0.5507;0 0 0 1];
matPos_cr_cl=inv(matPos_cl_cr);

% Manual selection ('clicks') of the edge extremities for matching
colork={'cyan' 'magenta' 'yellow' 'red' 'green' 'blue' 'black' 'cyan'};
%pour les 5 points qu'on rentre manuellement
%floor : rounds each element of X to the nearest integer less than or equal
to that element.
%point 1 : u1->y, v1->x
%point 2 : u2->y, v2->x
for p=1:N
    figure(2);                                          % Select the image
        [y,x]=  ginput(1);  u1(p)=floor(y);  v1(p)=floor(x);      %  Get  the
coordinates of the 1st click
```

```
        [y,x]=  ginput(1);  u2(p)=floor(y);  v2(p)=floor(x);      % Get the
coordinates of the 2nd click
                          text((u1(p)+u2(p))/2+10,       (v1(p)+v2(p))/2+10,
num2str(p),'color',colork{p});  % Show the number of edge
    line([u1(p) u2(p)],[v1(p) v2(p)],'color',colork{p});
      % Plot the line

    % ****************************************************************
      % Partie 3.3-1: Calcul de normale
      %   ******* À compléter ******* %
      %trouver les vecteurs directeurs du plan
      vector_segment = [u2(p)-u1(p), v2(p) - v1(p) , 0];%l1
      vector_origin_pt1= [(u1(p) - u0_1)/alphaU_1, (v1(p) - v0_1)/alphaV_1 ,
1];%l2

      %C = cross(A,B) returns the cross product of A and B.
    %If A and B are vectors, then they must have a length of 3.
              normale    =    cross(vector_origin_pt1,    vector_segment)    /
norm(cross(vector_origin_pt1, vector_segment))

      %normale(1)=0; normale(2)=0; normale(3)=0;    % Apres avoir complete,
commenter cette ligne

      App2d = [ App2d; normale(1) normale(2) normale(3) ];
end;

% Manual selection ('clicks') of the edges in the model
set(ll1, 'ButtonDownFcn', @button_down);
set(ll2, 'ButtonDownFcn', @button_down);
set(ll3, 'ButtonDownFcn', @button_down);
set(ll4, 'ButtonDownFcn', @button_down);
set(ll5, 'ButtonDownFcn', @button_down);
set(ll6, 'ButtonDownFcn', @button_down);
set(ll7, 'ButtonDownFcn', @button_down);
set(ll8, 'ButtonDownFcn', @button_down);
set(ll9, 'ButtonDownFcn', @button_down);
set(ll10, 'ButtonDownFcn', @button_down);
set(ll11, 'ButtonDownFcn', @button_down);
set(ll12, 'ButtonDownFcn', @button_down);

% Wait for the edges of the model to be selected
while size(App3d_ini, 1) ~= N
    pause(1)
end

% %%%%%%%%%%%% OPTIMIZATION %%%%%%%%%%%% %

%but trouver la bonne matrice de transfo du CAO vers la 2D à la fin
disp('Start optimization...');
```

```
soluL(1)=-2.1;       soluL(2)=0.7;       soluL(3)=2.7;       soluL(4)=151.6;
soluL(5)=46.8;soluL(6)=2164.8;
%la on a une solution qui est pas assez bonne
matPos=TransFromParam(soluL); % Homogeneous transformation
%matPos, matrice homogène = Mat_trans_hom*Mat_rot_hom*P_hom
%

%TransPoint : passe du repère CAO au repère camera (à check)
for p=1:N
                                        [App3d_mod(p,1)          App3d_mod(p,3)
App3d_mod(p,5)]=TransPoint(matPos,App3d_ini(p,1),App3d_ini(p,3),App3d_ini(p,
5));

[App3d_mod(p,2),App3d_mod(p,4),App3d_mod(p,6)]=TransPoint(matPos,App3d_ini(p
,2),App3d_ini(p,4),App3d_ini(p,6));
end;

% Shows a projection of the selected (clicked) edges according to the initial
pose value
mod_A = Trans_2_modA(N, App3d_ini);
figure(3); colormap(gray(256));
image(image_1);

DrawModel(alphaU_1,alphaV_1,u0_1,v0_1,mod_A,matPos); % 3D model drawing
oldCrit=CritEval(N,App2d,App3d_mod);              % Criteria evaluation
disp(['Initial criteria : ', num2str(oldCrit)]);
lambda= 0.001; iter = 0;
dummy = input('press a key to continue...');

while (1),
      J=zeros(2*N,6);
    lig=1;
    for p=1:N
      vectN(1)=App2d(p,1); vectN(2)=App2d(p,2); vectN(3)=App2d(p,3);
      [App3d_mod(p,1)                                          App3d_mod(p,3)
App3d_mod(p,5)]=TransPoint(matPos,App3d_ini(p,1),App3d_ini(p,3),App3d_ini(p,
5));

[dyda,y0]=GlobaleDerivative(vectN,App3d_mod(p,1),App3d_mod(p,3),App3d_mod(p,
5));
      J(lig,:)=dyda';
      L(lig,1)=-y0;
      lig=lig+1;

[App3d_mod(p,2),App3d_mod(p,4),App3d_mod(p,6)]=TransPoint(matPos,App3d_ini(p
,2),App3d_ini(p,4),App3d_ini(p,6));

[dyda,y0]=GlobaleDerivative(vectN,App3d_mod(p,2),App3d_mod(p,4),App3d_mod(p,
6));
      J(lig,:)=dyda';
      L(lig,1)=-y0;
```

```
      lig=lig+1;
    end;
    JJ=J'*J;
    for i=1:length(JJ);
      JJ(i,i)=JJ(i,i)*(1+lambda);
    end;

      % ********************************************************************
      % Partie 3.3-5: À COMPLÉTER AVEC LA FONCTION TransFromParam(.)******%

      % dsoluL = ??
      % dmatPos = TransFromParam(dsoluL);
      % matPos =
      dsoluL = inv(JJ) * J' * L;% cf cours
      dmatPos = TransFromParam(dsoluL);% les chagements à faire sur les
param
      matPos = dmatPos * matPos;%applique les transformations sur les param

      % ********************************************************************

      soluL=ParamFromTrans(matPos);%matrice de passage repère monde à repère
caméra
      for p=1:N
      [App3d_mod(p,1)                                          App3d_mod(p,3)
App3d_mod(p,5)]=TransPoint(matPos,App3d_ini(p,1),App3d_ini(p,3),App3d_ini(p,
5));

[App3d_mod(p,2),App3d_mod(p,4),App3d_mod(p,6)]=TransPoint(matPos,App3d_ini(p
,2),App3d_ini(p,4),App3d_ini(p,6));
    end;
    newCrit=CritEval(N,App2d,App3d_mod);
    if ((newCrit<EPS)|(abs(oldCrit-newCrit)<1e-10))
      break;
    end;

    % Show projection according to estimated pose at each iteration
    figure(3);colormap(gray(256));
    image(image_l);
    DrawModel(alphaU_1,alphaV_1,u0_1,v0_1,mod_A,matPos); % 3D model drawing
    disp(['Iteration [' num2str(iter) '] Error : ' num2str(newCrit) ]);
    iter = iter + 1;
    oldCrit = newCrit;
end;

disp(['6-uplet                solution                :                ',
num2str(soluL(1)),',',num2str(soluL(2)),',',num2str(soluL(3)),',',...
    num2str(soluL(4)),',',num2str(soluL(5)),',',num2str(soluL(6))]);
disp(['Error after convergence : ', num2str(newCrit)]);
figure(3); colormap(gray(256));
image(image_l); title 'Object left image (after convergence)';
```

```
DrawModel(alphaU_1,alphaV_1,u0_1,v0_1,modA,matPos);  % 3D model drawing in
left image

% ********************************************************************
% Partie 3.3-6: À COMPLÉTER POUR PROJETER DANS LA SECONDE IMAGE *** %

% matPos = ??

% ********************************************************************

figure(4); colormap(gray(256));
image(image_r); title 'Object right image (after convergence)';
DrawModel(alphaU_2,alphaV_2,u0_2,v0_2,modA,matPos);  % 3D model drawing in
left image



%% ------------------- FUNCTIONS ------------------------------ %%

function mod_A = Trans_2_modA(nbr, app3d)
mod_A = zeros(nbr, 6);
for i=1:nbr
    mod_A(i, 1:3) = [app3d(i,1) app3d(i,3) app3d(i,5)];
    mod_A(i, 4:end) = [app3d(i,2) app3d(i,4) app3d(i,6)];
end;


%animation du model 3D
%mod_A-> model
function DrawModel(alphaU,alphaV,u0,v0,modA,matPos)
%m = size(X,dim)
for p=1:size(modA, 1)

    % ********************************************************************
      % Partie 3.3-2: Compléter
      %    **** À COMPLÉTER. UTILISER LES FONCTIONS Projection et TransPoint
****
      %take point from CAO
    X1_CAO = modA(p, 1);
    Y1_CAO = modA(p, 2);
    Z1_CAO = modA(p, 3);
      [X1_mod,Y1_mod,Z1_mod] = TransPoint(matPos, X1_CAO, Y1_CAO, Z1_CAO);%
      [u1 v1] = Projection(alphaU,alphaV,u0,v0,X1_mod,Y1_mod,Z1_mod)

      X2_CAO = modA(p, 4);
    Y2_CAO = modA(p, 5);
    Z2_CAO = modA(p, 6);
      [X2_mod,Y2_mod,Z2_mod] = TransPoint(matPos, X2_CAO, Y2_CAO, Z2_CAO);%
      [u2 v2] = Projection(alphaU,alphaV,u0,v0,X2_mod,Y2_mod,Z2_mod)

      %u1=0; u2=0; v1=0; v2=0; % Apres avoir complete, commenter cette ligne
```

```
    % ********************************************************************


    line([u1 u2],[v1 v2],'color','red');
end;


%matrice de passage repère monde à repère caméra
function matPos=TransFromParam(soluL)
aa=soluL(1); bb=soluL(2); gg=soluL(3);
matPos=eye(4,4);
matPos(1,1)=cos(gg)*cos(bb);
matPos(1,2)=cos(gg)*sin(bb)*sin(aa)-sin(gg)*cos(aa);
matPos(1,3)=cos(gg)*sin(bb)*cos(aa)+sin(gg)*sin(aa);
matPos(1,4)=soluL(4);
matPos(2,1)=sin(gg)*cos(bb);
matPos(2,2)=sin(gg)*sin(bb)*sin(aa)+cos(gg)*cos(aa);
matPos(2,3)=sin(gg)*sin(bb)*cos(aa)-cos(gg)*sin(aa);
matPos(2,4)=soluL(5);
matPos(3,1)=-sin(bb);
matPos(3,2)=cos(bb)*sin(aa);
matPos(3,3)=cos(bb)*cos(aa);
matPos(3,4)=soluL(6);


function soluL=ParamFromTrans(matPos)
sb=-matPos(3,1);
if abs(sb) >=0.99999
    if sb >=0.0
      aMoinsg=atan2(matPos(2,2),matPos(1,2));
      soluL(1)=aMoinsg + soluL(3);
      soluL(2)=pi/2;
    else
      aPlusg=atan2(matPos(2,2),-matPos(1,2));
      soluL(1)=aPlusg-soluL(3);
      soluL(2)=-pi/2;
      end;
else
      cb=sqrt(1.0-sb*sb);
      sa=matPos(3,2)/cb;
      ca=matPos(3,3)/cb;
      sg=matPos(2,1)/cb;
      cg=matPos(1,1)/cb;
      soluL(1)=atan2(sa,ca);
      soluL(2)=atan2(sb,cb);
      soluL(3)=atan2(sg,cg);
end;
if soluL(1)>pi soluL(1)=soluL(1)-2*pi; end;
if soluL(2)>pi soluL(2)=soluL(2)-2*pi; end;
if soluL(3)>pi soluL(3)=soluL(3)-2*pi; end;
soluL(4)=matPos(1,4); soluL(5)=matPos(2,4); soluL(6)=matPos(3,4);
```

```
%matPos, matrice homogène = Mat_trans_hom*Mat_rot_hom*P_hom
%return X,Y,Z
function [X_mod Y_mod Z_mod]=TransPoint(matPos,X_ini,Y_ini,Z_ini)
X_mod=matPos(1,1)*X_ini    +    matPos(1,2)*Y_ini    +    matPos(1,3)*Z_ini    +
matPos(1,4);
Y_mod=matPos(2,1)*X_ini    +    matPos(2,2)*Y_ini    +    matPos(2,3)*Z_ini    +
matPos(2,4);
Z_mod=matPos(3,1)*X_ini    +    matPos(3,2)*Y_ini    +    matPos(3,3)*Z_ini    +
matPos(3,4);


%projection image du model; 3D to 2D
function [u v]=Projection(alphaU,alphaV,u0,v0,X,Y,Z)
% ********************************************************************
% Partie 3.3-2: Compléter
%   ******* À COMPLÉTER ******* %

u = alphaU*X/Z + u0;
v = alphaV*Y/Z + v0;


%u=0; v=0;   % Après, commenter cette ligne

%App2d : normals to the segment selected previously
%App3d_mod : model CAO in camera
function [crit]=CritEval(N,App2d,App3d_mod)
err=0;
    for p=1:N;
      % ************************************************************
      % Partie 3.3-3: Compléter
      %   ******* À COMPLÉTER (POUR ERR)******* %
      %add the normal * the segment (=0 good matrix intrinsic)
      err_1    =    App2d(p,:)    *    [App3d_mod(p,1)    ;    App3d_mod(p,3)    ;
App3d_mod(p,5)];
      err_2    =    App2d(p,:)    *    [App3d_mod(p,2)    ;    App3d_mod(p,4)    ;
App3d_mod(p,6)];
      err = err + err_1*err_1 + err_2*err_2;
    end;
crit=sqrt(err/2*N);


function [dyda,y0]=GlobaleDerivative(vectN,X,Y,Z)
% ********************************************************************
% Partie 3.3-4: Compléter
%   ******* Á COMPLÉTER (MODIFIER) ******* %
%voir question 1.2.2
 dyda(1,1)= vectN * [0; -Z ; Y];
 dyda(2,1)= vectN * [Z ; 0 ; -X];
 dyda(3,1)= vectN * [-Y ; X ; 0];
```

```
 dyda(4,1)= vectN(1);
 dyda(5,1)= vectN(2);
 dyda(6,1)= vectN(3);

% ********************************************************************
y0=vectN(1)*X + vectN(2)*Y + vectN(3)*Z;
```