TP Visual servoing

Fait par:

- BELLAHCENE Kheir Eddine
- GEROL Logan

1. Définir une caméra avec les paramètres par default de la Toolbox. La fonction à utliser est CentralCamera

```
clc, clear, close all
cam =  CentralCamera('default')
```

```
principal point not specified, setting it to centre of image plane

cam =

name: default [central-perspective]
  focal length:   0.008
  pixel size:     (1e-05, 1e-05)
  principal pt:   (512, 512)
  number pixels:  1024 x 1024
  pose:           t = (0, 0, 0), RPY/yxz = (0, 0, 0) deg
```

2. Afficher les paramètres intrinsèques et extrinsèques de la caméra.

```
% intrinsque de la caméra
 K=cam.K
```

```
K = 3×3
   800     0   512
     0   800   512
     0     0     1
```

```
 % extrinsque de la caméra
 T=cam.T % intrinsque de la caméra
```

```
T =
         1         0         0         0
         0         1         0         0
         0         0         1         0
         0         0         0         1
```

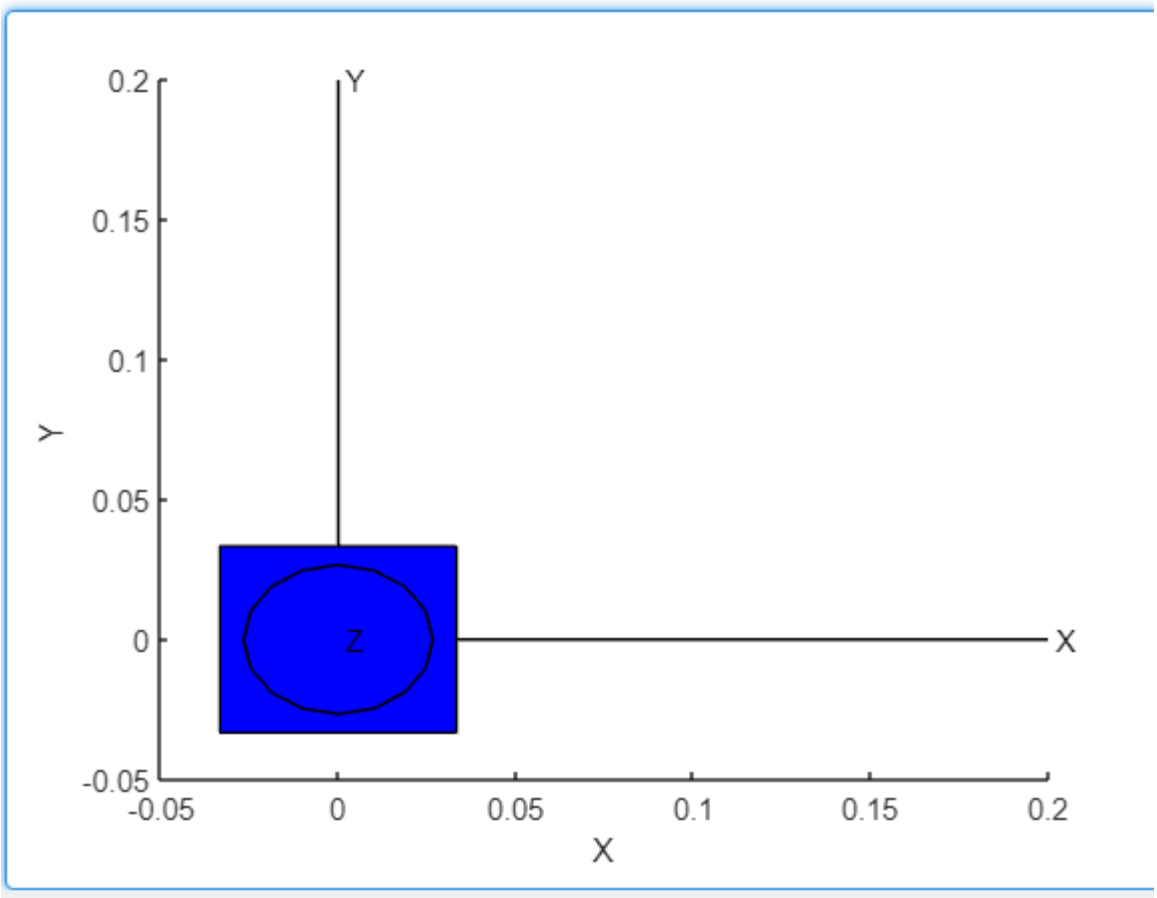
2.2 La matrice intrinsque et extrinsique

```
TK=cam.C
```

```
TK = 3×4
   800     0   512     0
     0   800   512     0
     0     0     1     0
```

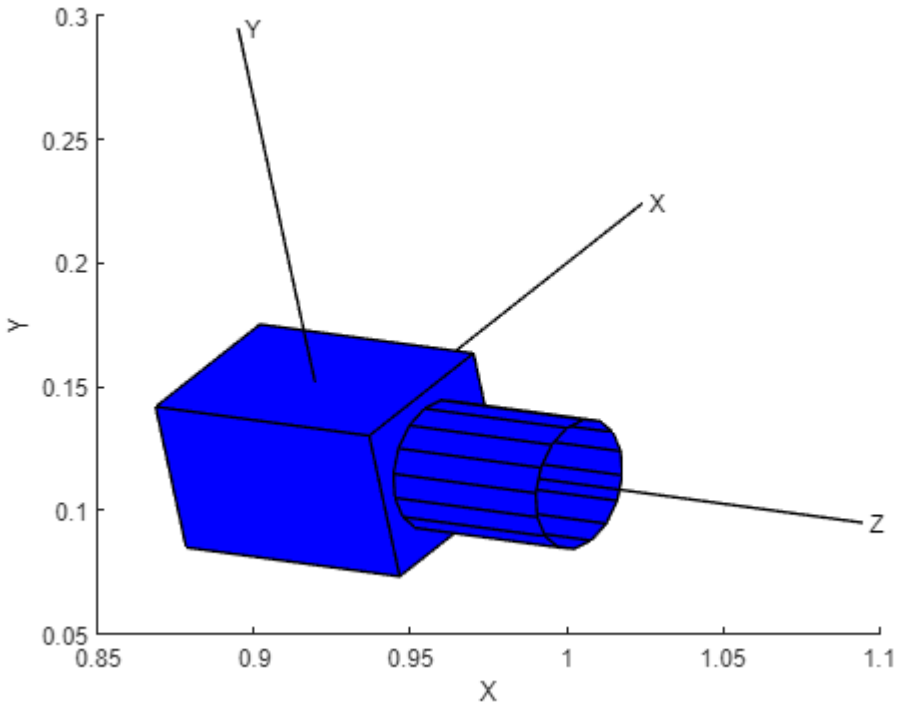
3. Afficher une caméra dans un repère 3D en u<lisant la fonction plot_camera de la classe CentralCamera.

```
figure;
cam.plot_camera()
```



4. Modifier la pose de la caméra via son paramètre T et réafficher la caméra en 3D. Vous pouvez utliser la ligne suivante pour définir une pose via une matrice de transformation homogène :

```
theta_x = pi/4;
theta_y = pi/4;
theta_z = pi/4;
tx = 0.2;
ty = 0.2;
tz = 1;
Tc1 = trotz(theta_z)*troty(theta_y)*trotx(theta_x)*transl(tx,ty,tz);
cam.T=Tc1;
% Attribution de cette matrice de transformation homogène à notre caméra
% pour modifier la pose de notre caméra par la pose déclaré
```



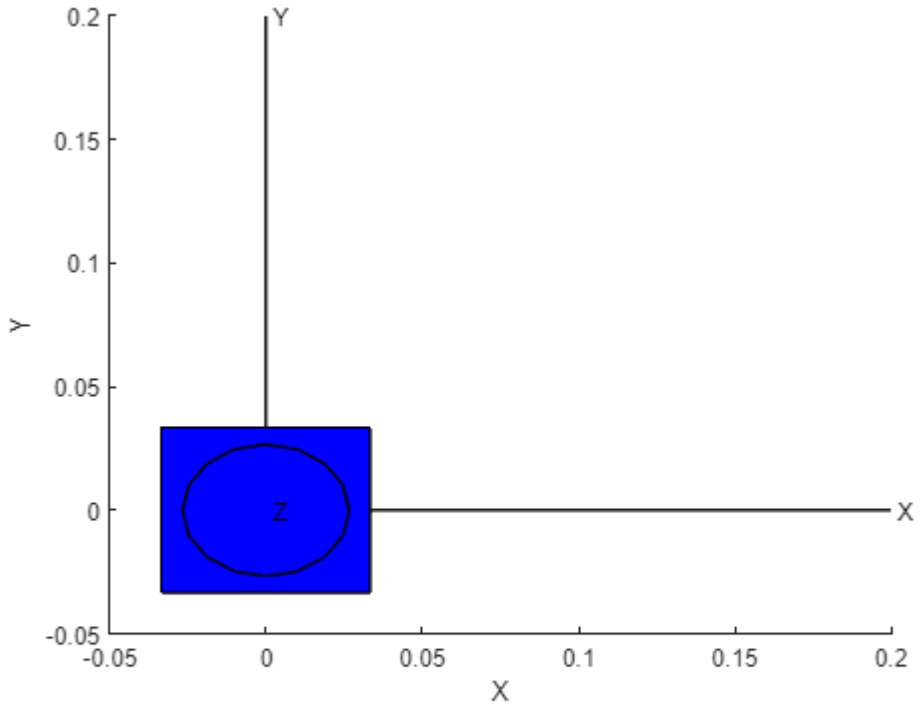
5. Définir un ensemble de points 3D dans le repère objet ayant une translation de 3m selon l'axe Z, par rapport au repère monde. On notera ceQe pose To et on u<lisera la ligne de code Matlab suivante pour générer les points 3D : P = mkgrid(2, 0.5, 'pose', To);

```
% Définition des Points 3D dans le repère objet avec pose To
To = transl(0,0,3);
P = mkgrid(2, 0.5, 'pose', To);
```

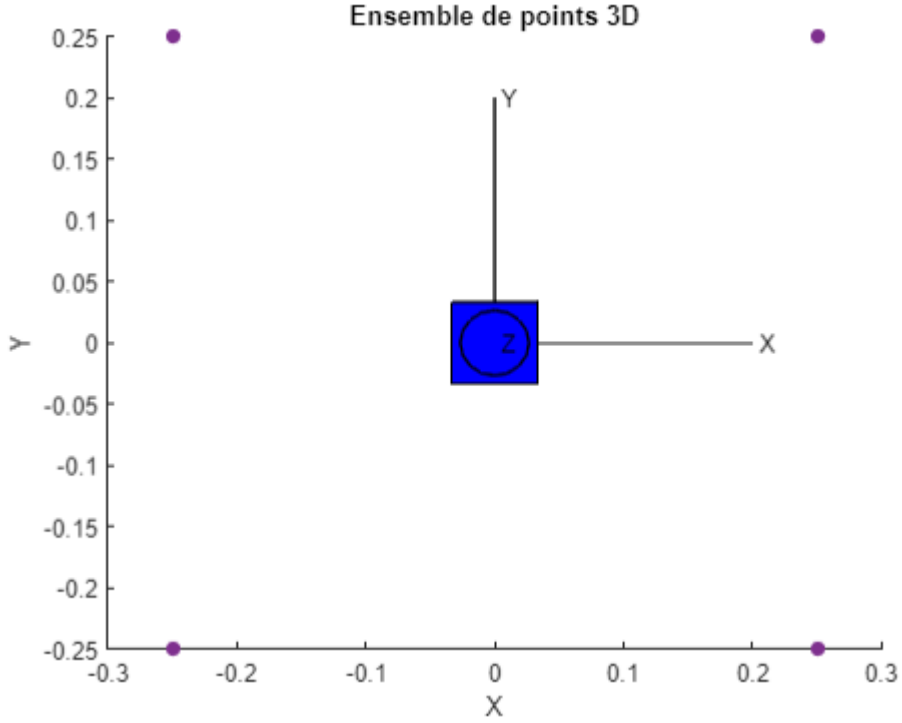
6. Définir une pose de la caméra qu'on appellera Tc1 et qui est confondue avec le repère monde.

```
%POSE DESIREE
theta_x1 = 0;
theta_y1 = 0;
theta_z1 = 0;
tx = 0;
ty = 0;
tz = 0;
Tc1 = trotz(theta_z1)*troty(theta_y1)*trotx(theta_x1)*transl(tx,ty,tz);
cam.T=Tc1;
hold on

% P(1,:) --> (les X des 4 points), P(2,:)--> (les Y des 4 points) et P(3,:) --> (les Z des 4 po
% scatter3 Sont de taille (1*4) et on a 4 points 3D definie par mkgrid
% sont les suivant:

% (     X  ,      Y  , Z)
% ( -0.2500,  -0.2500, 3)
% ( -0.2500,   0.2500, 3)
% (  0.2500,   0.2500, 3)
```

```
% (  0.2500,  -0.2500, 3)

% 'filled' pour les points seront remplie
% 'k' pour la coleur bleu
scatter3(P(1, :), P(2, :), P(3, :), 'filled');


% donner des titres au différentes axes et au figure
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Ensemble de points 3D')
hold off
```




```
cam =

name: default [central-perspective]
  focal length:   0.008
  pixel size:     (1e-05, 1e-05)
  principal pt:   (512, 512)
  number pixels:  1024 x 1024
  pose:           t = (0, 0, 0), RPY/yxz = (0, 0, 0) deg
```



7. Définir une seconde pose de la caméra qu'on appellera Tc2, ayant une rotation de pi/5 selon l'axe Z, et une translation de (1 1.2 0)T

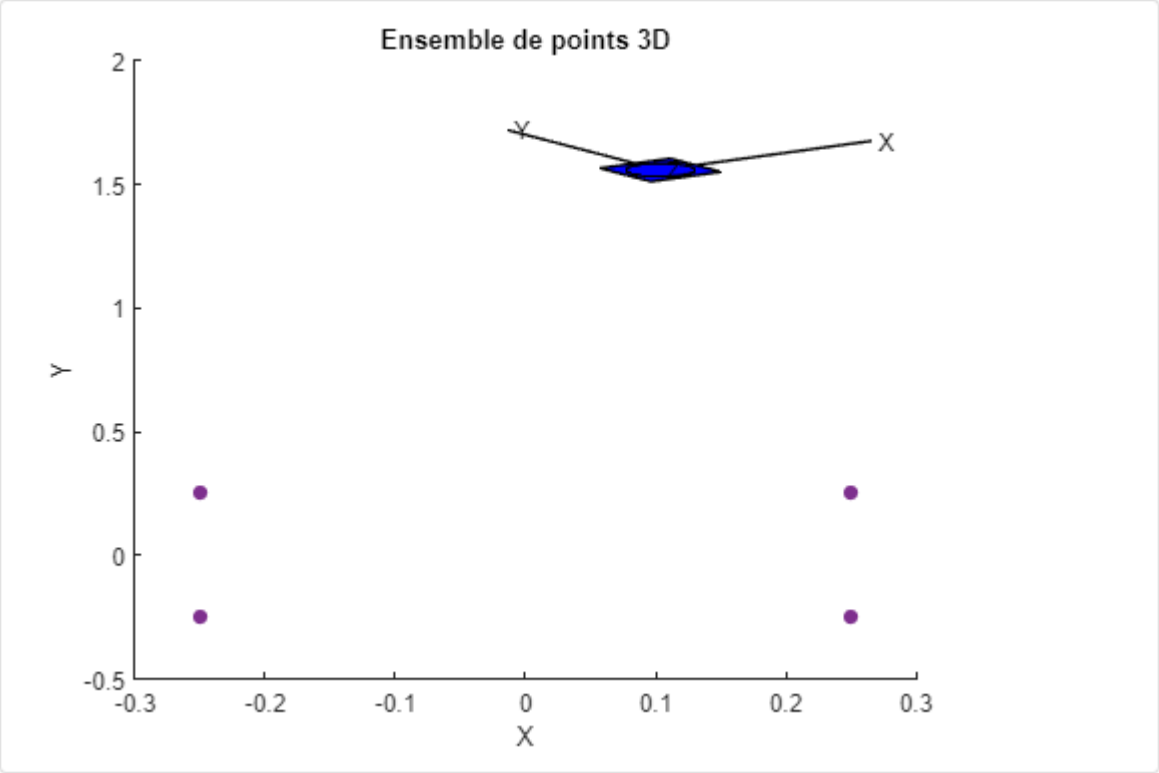
```
%POSE ACTUELLE
theta_x2 = 0;
theta_y2 = 0;
theta_z2 = pi/5;
tx2 = 1;
ty2 = 1.2;
tz2 = 0;
Tc2 = trotz(theta_z2)*troty(theta_y2)*trotx(theta_x2)*transl(tx2,ty2,tz2);
cam.T=Tc2
```

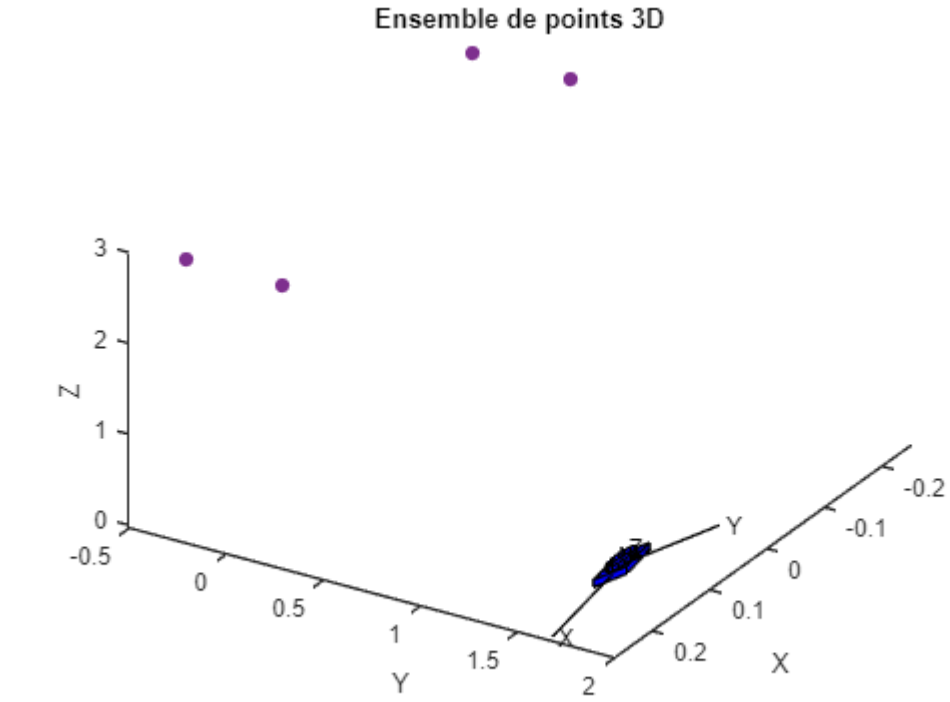
```
cam =

name: default [central-perspective]
  focal length:   0.008
  pixel size:     (1e-05, 1e-05)
  principal pt:   (512, 512)
  number pixels:  1024 x 1024
  pose:           t = (0.104, 1.56, 0), RPY/yxz = (36, 0, 0) deg
```

```
hold on
```

8. Utliser la fonction plot de la classe CentralCamera, pour projeter les points 3D définis précédemment et afficher leurs points images. Les deux poses de la caméra Tc1 et Tc2 (avec les op<ons de couleur rouge pour l'image à la pose Tc1 et bleu pour l'image à la pose Tc2, et de MarkerSize de 15) sont à considérer. Il est possible d'utliser l'option CentralCamera.hold pour afficher les points image sur la même figure.

```
%Afficher les points 3D dans le repère objet
scatter3(P(1, :), P(2, :), P(3, :), 'filled');

% donner des titres au différentes axes et au figure
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Ensemble de points 3D')
```





Code

```
view([121.33 42.00])
```

9. Afficher les coordonnées des points image en pixel (projetés dans les deux poses de la caméra) et générer/afficher les coordonnées normalisées de ces points ()voir le help de la fonction CentralCamera.normalized).

```
% On utilise la méthode plot pour projeter les points 3D dans plan image et
% afficher leurs points images en retournant les cordonnées pixel [u,v] de
```

```
% chaque points 3D et selon les deux pose

% les cordonées en pixel uv1 (2*4) des 4 points 3D selon la pose Tc1 Pose Désiré
% en rouge et avec un size de 30.
uv1 = cam.plot(P,'pose', Tc1,'.r','MarkerSize',30)
```

```
creating new figure for camera
h =
  Axes with properties:

            XLim: [0 1024]
            YLim: [0 1024]
          XScale: 'linear'
          YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Show all properties
make axes
uv1 = 2×4
  445.3333  445.3333  578.6667  578.6667
  445.3333  578.6667  578.6667  445.3333
```

```
% Maintenir le plan image pour afficher les 8 point (4 points selon les
% deux pose de la caméra) dans un seul plan image.
cam.hold

% les cordonées en pixel uv1 (2*4) des 4 points 3D selon la pose Tc2 Pose Actuelle
% en bleu et avec un size de 30.
uv2= cam.plot(P,'pose', Tc2,'.b','MarkerSize',30)
```

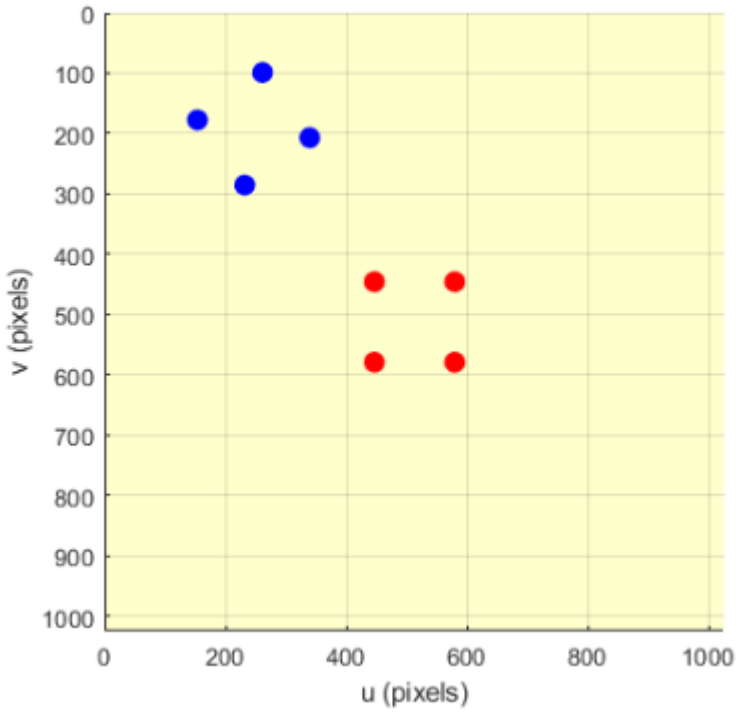
```
uv2 = 2×4
  152.2132  230.5846  338.4535  260.0821
  177.2512  285.1201  206.7488   98.8799
```

```
% Affichant les coordonées normalise de ces points images en pixel projetes
% dans les deux pose de la caméra

% [u,v] normaliser selon Tc1 Pose Désiré
uv1_norm = cam.normalized(uv1)
```

```
uv1_norm = 2×4
   -0.0833   -0.0833    0.0833    0.0833
   -0.0833    0.0833    0.0833   -0.0833
```

```
% [u,v] normaliser selon Tc2 Pose Actuelle
uv2_norm = cam.normalized(uv2)
```

```
uv2_norm = 2×4
   -0.4497   -0.3518   -0.2169   -0.3149
   -0.4184   -0.2836   -0.3816   -0.5164
```



Déplacement de la caméra

Voici un exemple de modification de la pose de la caméra en utilsant un mouvement composé à l'aide du torseur cinématique nommé vc.

$Vc=[vx\ vy\ vz\ \omega x\ \omega y\ \omega z]T$

où vx,vy,vz sont les composantes de la vitesse linéaire le long des axes X, Y et Z respectivement, et ωx,ωy,ωz sont les composantes de la vitesse angulaire autour des axes X, Y et Z respectivement.

Dans cet exemple nous avons donc une combinaison de translation autour l'axe x et z (vitesse linéaire de 2 unité par unité de temps autour de l'axe x et vitesse linéaire de 1 unité par unité de temps autour de l'axe z) ainsi que d'une rotation autour de l'axe z (vitesse angulaire de pi/5 radian par unité de temps autour de l'axe x).

```
%Initialisation
%
%dt = 0.02;
% vc = [2 0 1 0 0 pi/5];
% for i=1:50
%   cTc = trnorm(eye(4)*delta2tr(vc*dt));
%   Tc2 = Tc2 * cTc;
%   cam.T = Tc2;
%   cam.plot_camera;
%   pause(1)
%
% end
```

Schéma d'AV 2D

1. Créer une fonc<on nommée Ls qui prend comme paramètres, une matrice contenant les coordonnées de points images normalisées, et un vecteur contenant la profondeur (l'élément Z) de chaque point. La fonc<on renvoie la matrice d'interac<on pour un vecteur de primitives visuelles de type points 2D normalisés contenant les éléments x et y de chaque point.

```
function L = ls(md,Z_ones)

N = size(md, 2);

L = zeros(2*N, 6);

for i=1:N
    x = md(1, i);
    y = md(2, i);
    Z = Z_ones(i);

    L(2*i-1,:)=[-1/Z, 0, x/Z, x*y, -(1+x^2), y];
    L(2*i,:)=[0, -1/Z, y/Z, 1+y^2, -x*y, -x];

end
```

2. Proposer un programme de simula<on d'AV 2D ramenant la caméra de la pose Tc2 à la pose Tc1. Utliser les profondeurs exactes des points 3D dans le calcul de la matrice d'interaction.

```
close all
Zones=ones(1,size(P,2));
LTc1 = ls(uv1_norm,Zones)
```

```
LTc1 = 8×6
   -1.0000         0   -0.0833    0.0069   -1.0069   -0.0833
         0   -1.0000   -0.0833    1.0069   -0.0069    0.0833
   -1.0000         0   -0.0833   -0.0069   -1.0069    0.0833
         0   -1.0000    0.0833    1.0069    0.0069    0.0833
   -1.0000         0    0.0833    0.0069   -1.0069    0.0833
         0   -1.0000    0.0833    1.0069   -0.0069   -0.0833
   -1.0000         0    0.0833   -0.0069   -1.0069   -0.0833
         0   -1.0000   -0.0833    1.0069    0.0069   -0.0833
```

```
LTc2 = ls(uv2_norm,Zones)
```

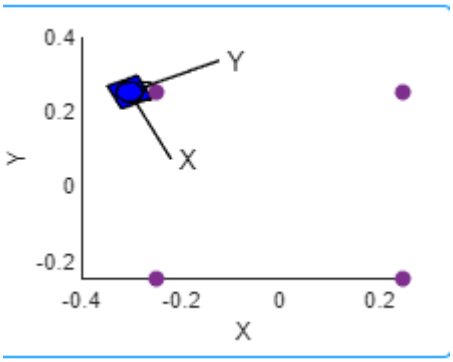
```
LTc2 = 8×6
   -1.0000         0   -0.4497    0.1882   -1.2023   -0.4184
         0   -1.0000   -0.4184    1.1751   -0.1882    0.4497
   -1.0000         0   -0.3518    0.0998   -1.1237   -0.2836
         0   -1.0000   -0.2836    1.0804   -0.0998    0.3518
   -1.0000         0   -0.2169    0.0828   -1.0471   -0.3816
         0   -1.0000   -0.3816    1.1456   -0.0828    0.2169
   -1.0000         0   -0.3149    0.1626   -1.0992   -0.5164
         0   -1.0000   -0.5164    1.2667   -0.1626    0.3149
```

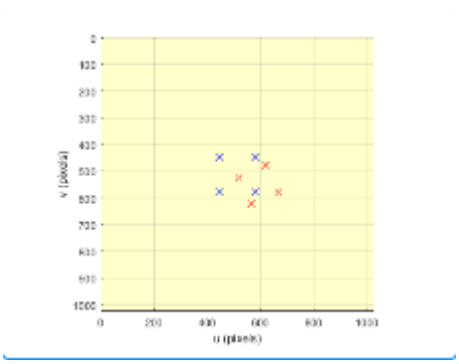
```
%s1=uv_norm(:);
delta=uv2_norm-uv1_norm;
%vc = [0 1 0 pi/5 0 0];
```

```
lambda=0.2;
dt=0.3;


 for i=1:50
     % Calculer l'erreur visuelle
     delta = uv2_norm - uv1_norm;

     % Calculer la matrice d'interaction
     L = ls(uv2_norm, ones(1, size(P, 2)));

     % Calculer la vitesse de la caméra
     vc = -lambda * pinv(L) * delta(:);

     cTc = trnorm(eye(4)*delta2tr(vc*dt));
     Tc2 = Tc2 * cTc;
     cam.T = Tc2;
     cam.plot_camera();
     hold on
     scatter3(P(1, :), P(2, :), P(3, :), 'filled');
     pause(0.1)

 end
cam.plot(P, 'pose', Tc1, 'xb', 'MarkerSize', 8)
```

```
creating new figure for camera
h =
  Axes with properties:

            XLim: [0 1]
            YLim: [0 1]
          XScale: 'linear'
          YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Show all properties
make axes
```

```
cam.hold
cam.plot(P, 'pose', Tc2, 'xr', 'MarkerSize', 8)
```





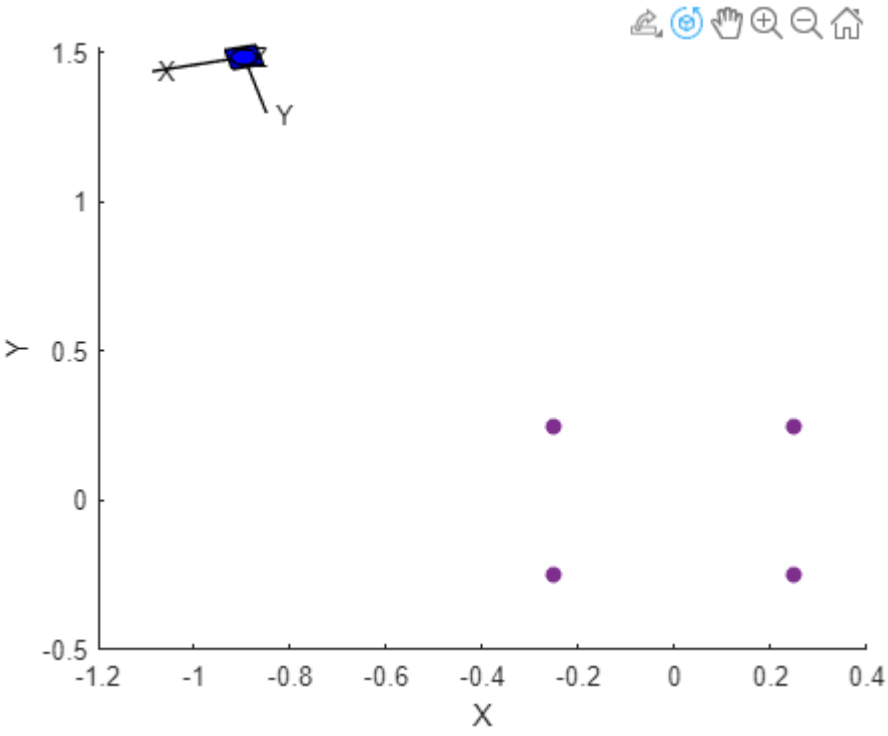
Dans cette simulation, la caméra se déplace de la pose `Tc2` à la pose `Tc1` en utilisant les profondeurs exactes des points 3D pour calculer la matrice d'interaction. Cela simule une situation où les profondeurs sont correctement estimées. Les performances dépendent de la précision des paramètres intrinsèques et extrinsèques de la caméra ainsi que de la qualité de l'estimation des profondeurs.

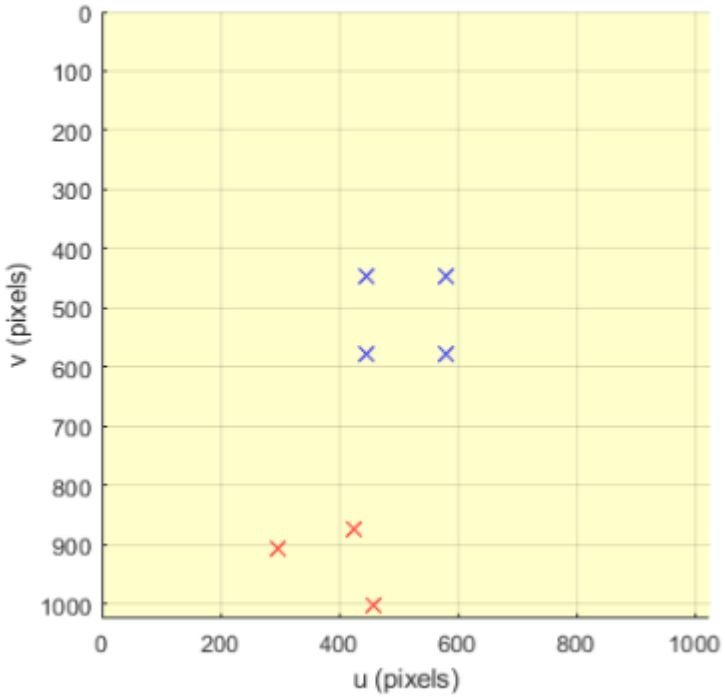
4. Réaliser une simulation d'AV2D en utilisant la matrice d'intéraction calculée à l'équilibre, c'est-à-dire calculée avec les données relatives à la position désirée de la caléra (la pose Tc1 ici).

```
close all


 for i=1:50
     % Calculer l'erreur visuelle
     delta = uv2_norm - uv1_norm;

     % Calculer la matrice d'interaction
     L = LTc1;

     % Calculer la vitesse de la caméra
     vc = -lambda * pinv(L) * delta(:);

     cTc = trnorm(eye(4)*delta2tr(vc*dt));
     Tc2 = Tc2 * cTc;
     cam.T = Tc2;
     cam.plot_camera();
     hold on
     scatter3(P(1, :), P(2, :), P(3, :), 'filled');
     pause(0.1)

 end
cam.plot(P, 'pose', Tc1, 'ob', 'MarkerSize', 8)
```

```
creating new figure for camera
h =
  Axes with properties:

            XLim: [0 1]
            YLim: [0 1]
          XScale: 'linear'
          YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Show all properties
make axes
```

```
cam.hold
cam.plot(P, 'pose', Tc2, 'xr', 'MarkerSize', 8)
```





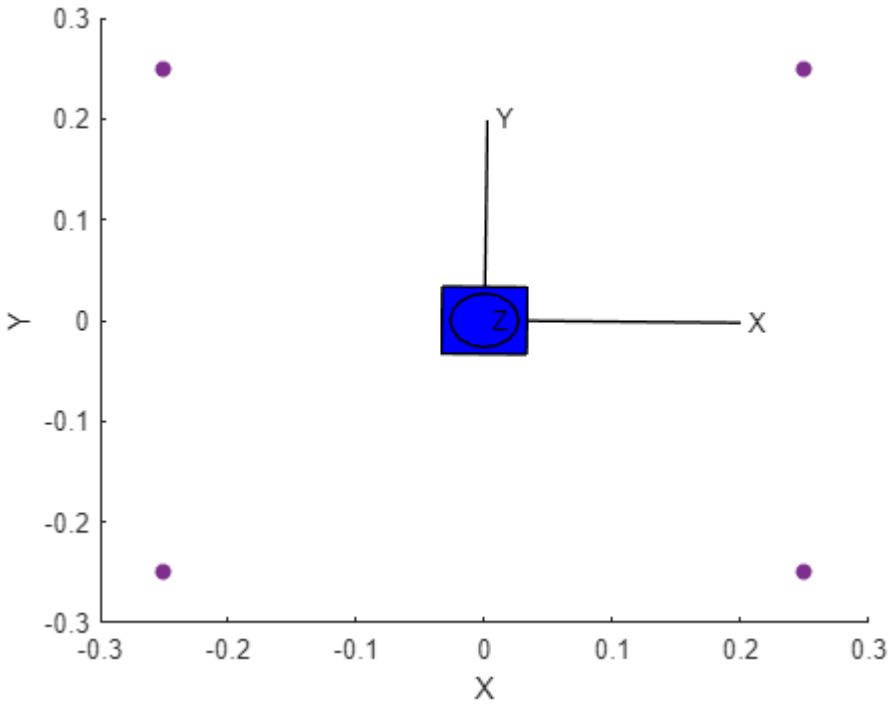
5. Modifier la pose initale de la caméra (Tc2) pour ne réaliser qu'une rotation selon l'axe Z de la caméra entre les deux poses Tc1 et Tc2 (rotation pure selon l'axe Z entre les deux poses caméras initale et désirée). Simuler avec les deux calculs de la matrice d'interaction (à chaque itération et à l'équilibre).

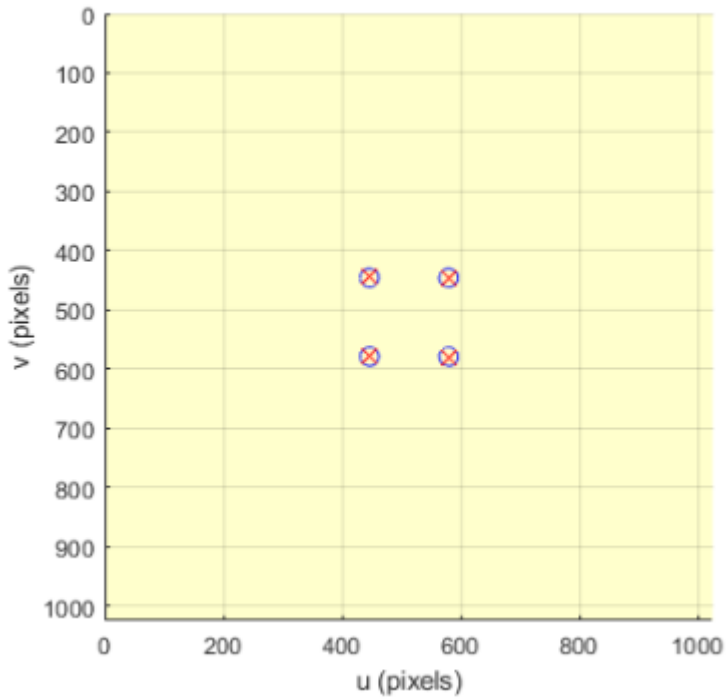
```
close all
% Modification  la pose initiale de la caméra pour une rotation pure autour de l'axe Z
theta_z_ini = 0; % Angle initial
Tc2_ini = trotz(theta_z_ini); % Pose initiale avec rotation pure autour de l'axe Z
cam.T = Tc2_ini;
```

```
% Calcul de la matrice d'interaction à l'équilibre (utilisant Tc1)
L_eq = ls(uv1_norm, ones(1, size(P, 2)));

dt = 0.1;
lambda = 0.2;

for i = 1:50
    % Calcul de l'erreur visuelle
    delta = uv2_norm - uv1_norm;

    % Utilisation de la matrice d'interaction à l'équilibre
    L = L_eq;

    % Calcul de la vitesse de la caméra
    vc = -lambda * pinv(L) * delta(:);

    % Mise à jour la pose de la caméra avec une rotation pure autour de l'axe Z
    theta_z_update = vc(6) * dt;
    Tc2 = trotz(theta_z_update);
    cam.T = Tc2;
    cam.plot_camera();
    hold on
    scatter3(P(1, :), P(2, :), P(3, :), 'filled');
    pause(0.1)

end
cam.plot(P, 'pose', Tc1, 'ob', 'MarkerSize', 8)
```

```
creating new figure for camera
h =
  Axes with properties:

            XLim: [0 1]
            YLim: [0 1]
          XScale: 'linear'
          YScale: 'linear'
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Show all properties
make axes
```

```
cam.hold
cam.plot(P, 'pose', Tc2, 'xr', 'MarkerSize', 8)
```



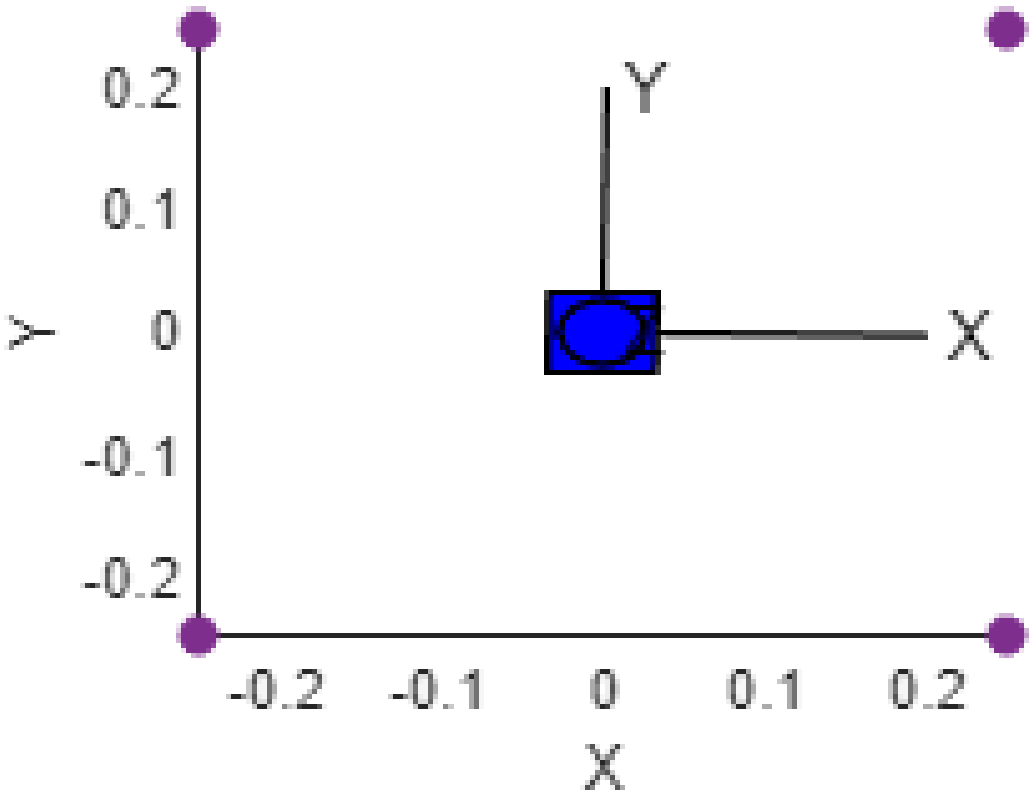


Grâce à cette application nous pouvons observer que la rotaion pure autour de l'axe z est une solution qui peut être retenu car l'objectif à été rempli.

6. Simuler l'AV2D en utlisant la moyenne des deux matrices d'interaction calculées dans la question précédente : L = 0.5* (L1+L2).

```
close all
L_average = 0.5 * (LTc1 + LTc2);
```

```
for i=1:50
    % Calcul de l'erreur visuelle
    delta = uv2_norm - uv1_norm;

    % Calcul de la matrice d'interaction
    L = L_average;

    % Calcul de la vitesse de la caméra
    vc = -lambda * pinv(L) * delta(:);

    theta_z_update = vc(6) * dt;
    Tc2 = trotz(theta_z_update);
    cam.T = Tc2;
    cam.plot_camera();
    hold on
    scatter3(P(1, :), P(2, :), P(3, :), 'filled');
    pause(0.1)

end
```

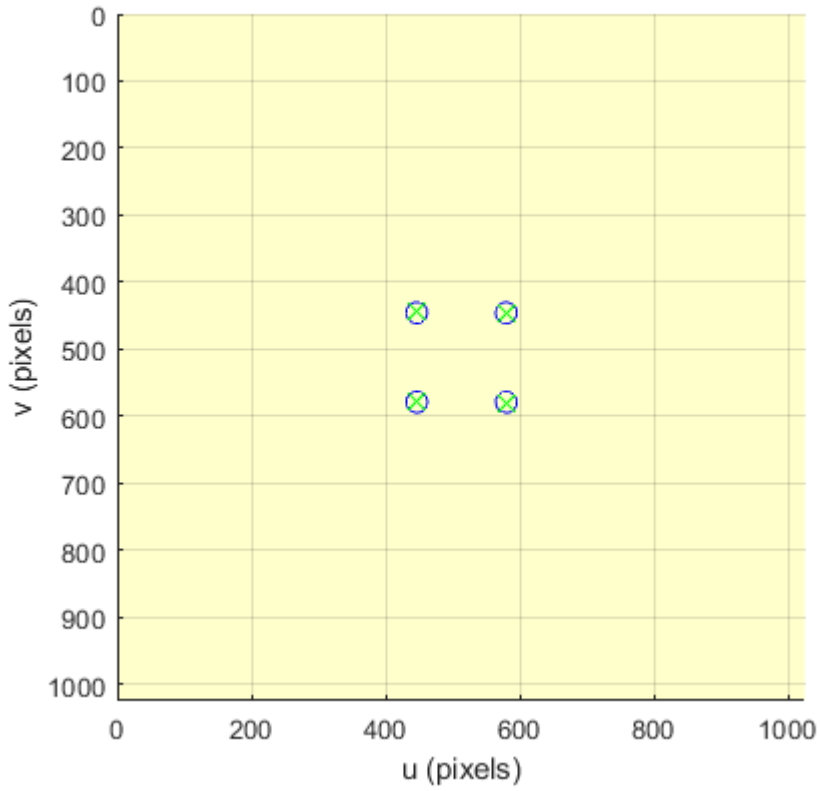


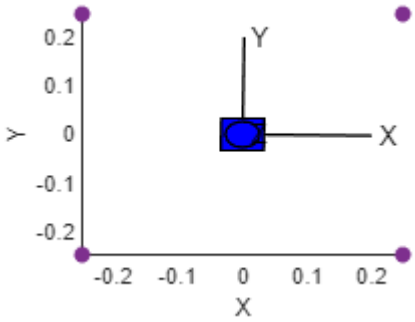

```
cam.plot(P, 'pose', Tc1, 'ob', 'MarkerSize', 8)
```

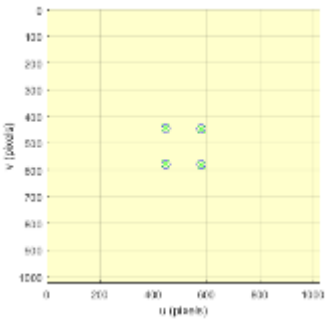
```
creating new figure for camera
h =
  Axes with properties:

            XLim: [0 1]
            YLim: [0 1]
          XScale: 'linear'
          YScale: 'linear'
```

```
    GridLineStyle: '-'
         Position: [0.1300 0.1100 0.7750 0.8150]
            Units: 'normalized'

  Show all properties
make axes
```

```
cam.hold
cam.plot(P, 'pose', Tc2, 'xg', 'MarkerSize', 8)
```







Cette approche est un compromis entre les deux matrices d'interaction et peut potentiellement améliorer la robustesse du système face à des variations imprévues.Nous pouvons le constater car lors de l'application nous sommes arrivé à placer la caméra dans la pose désirée.

CONCLUSION : Ce TP sur l'asservissement visuel 2D a permis d'explorer et de mettre en pratique divers concepts et techniques clés dans le domaine de la vision par ordinateur. À travers une série de simulations, nous avons étudié différentes situations de déplacement de la caméra et évalué l'impact de divers facteurs sur les performances du système. Chaque simulation a offert des perspectives uniques sur les performances et la robustesse du système d'asservissement visuel.