

09/01/2022

# Compte Rendu du Projet de Vision



BENAISSE Tarek, BENBRAHIM Mohamed El  
**Amine**  
M1 ISI

## Introduction :

Dans le cadre de l'UE Vision par ordinateur et pour appliquer toutes nos connaissances acquises pendant ce semestre, on a réalisé ce projet qui consiste à générer un objet 3D dans une vidéo qu'on a filmé avec un smartphone.

Ceci nécessite de faire la calibration de la caméra, donc le calcul de la matrice intrinsèque et la pose, puis un calcul d'homographie afin de trouver les matrices de projection pour mapper les points 3D de notre objet vers chaque image de la vidéo qu'on a filmé, à la fin on reforme la même vidéo avec notre objet 3D qui apparaît au-dessus.

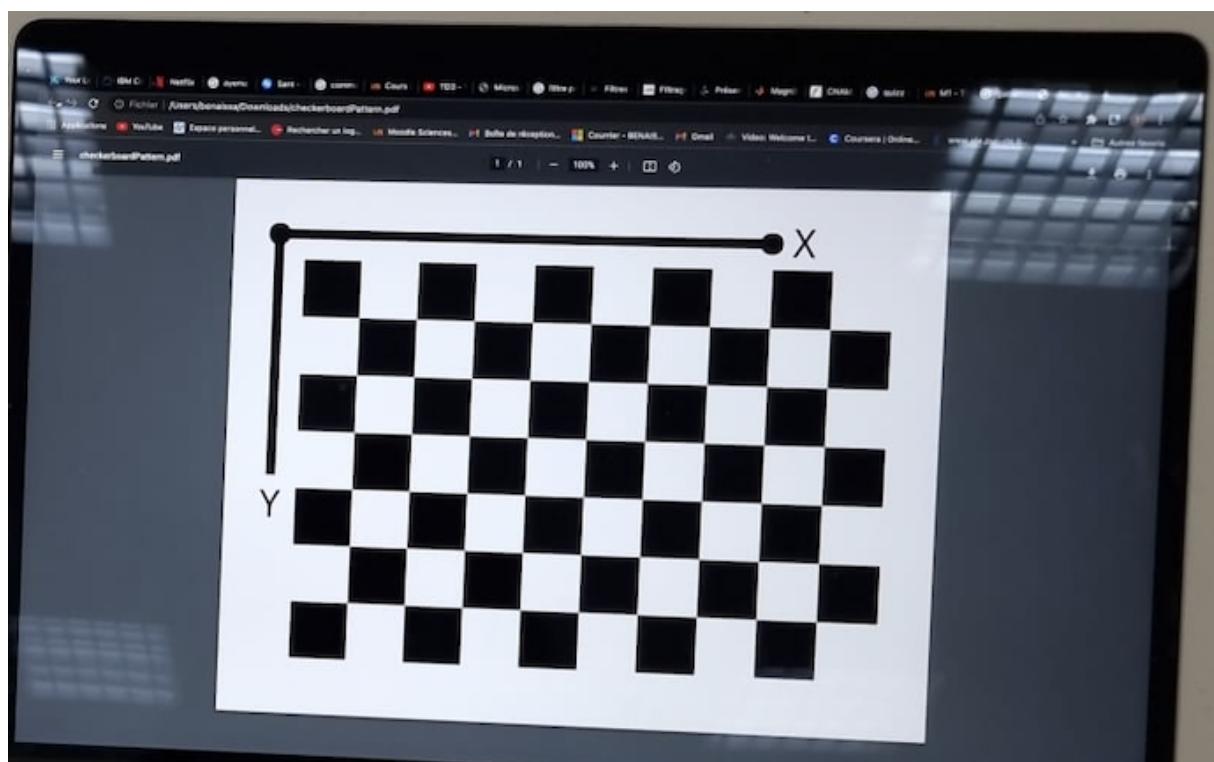
## Les méthodes :

### 1) Calibration de la caméra :

La première étape consiste à calibrer la caméra, c'est-à-dire trouver sa matrice de projection  $P$  qui permet d'aller d'un point dans le repère monde  $(x,y,z)$  vers le point image  $(u,v)$ .

Pour cette étape on a utilisé la Toolbox « Image processing and computer vision » qui permettent de calibrer la caméra à partir d'une séquence des images d'une mire (échiquier) prises avec différents angles et positionnements.

On a utilisé les images d'échiquier prise avec le smartphone (voir exemple : image ci-dessous), qu'on a chargé sur Matlab, puis on a calibré la caméra à l'aide de la fonction « CameraCalibrator » qui renvoie plusieurs paramètres de la caméra dont la matrice intrinsèque qui est la même pour toutes les images, car elle ne dépend que de l'appareil utilisé pour prendre les photos (notamment la focale, les dimensions et forme des pixels ...), on l'a enregistré dans notre Workspace et on a l'a utilisé pour calculer de nous-même la matrice extrinsèque (la pose de la camera spécifique à chaque image) en utilisant la DLT et une méthode dérivée de Zhang's method.



Exemple d'image utilisée en calibration

Pour tester le bon fonctionnement de notre calibration, on a utilisé deux méthodes :

### Méthode 1 :

On a pris quelques point world de cameraParams, puis on a calculé leurs projections avec nos propres matrices de projection puis on a comparé avec les résultats fournis par cameraParams.

Le code de cette partie est présenté ci-dessous :

```
K=cameraParams.IntrinsicMatrix'; %recuperation du matrice intrinseque
WorldPoints=cameraParams.WorldPoints; %recuperation des points monde
ReprojectedPoints=cameraParams.ReprojectedPoints; %recuperation des points image
Rot=cameraParams.RotationMatrices(:,:,13)'; %les matrices de rotation img10
Trans=cameraParams.TranslationVectors(13,:); %les vecteurs de translation img10
P=K*[Rot(:,:,1),Rot(:,:,2),Trans]; %la matrice de projection P
WorldPoint=[cameraParams.WorldPoints,ones(size(cameraParams.WorldPoints,1),1)]';
ReprojPoint=P*WorldPoint; %recalcul des points images à partir des points monde et matrice
ReprojPoint=ReprojPoint./ReprojPoint(3,:); %normalisation des points reprojetés
ReprojectedPoints=ReprojectedPoints(:,:,13)'; %les vrais points images pour comparer
```

Les points images données par cameraParams

```
ReprojectedPoints =
1.0e+02 *
Columns 1 through 5
6.225487071131997 6.233194307390839 6.240740209512845 6.248115278418231 6.255312664796162
3.442252708113126 3.908505223160574 4.365826659390779 4.814187483660247 5.253899977247213
```

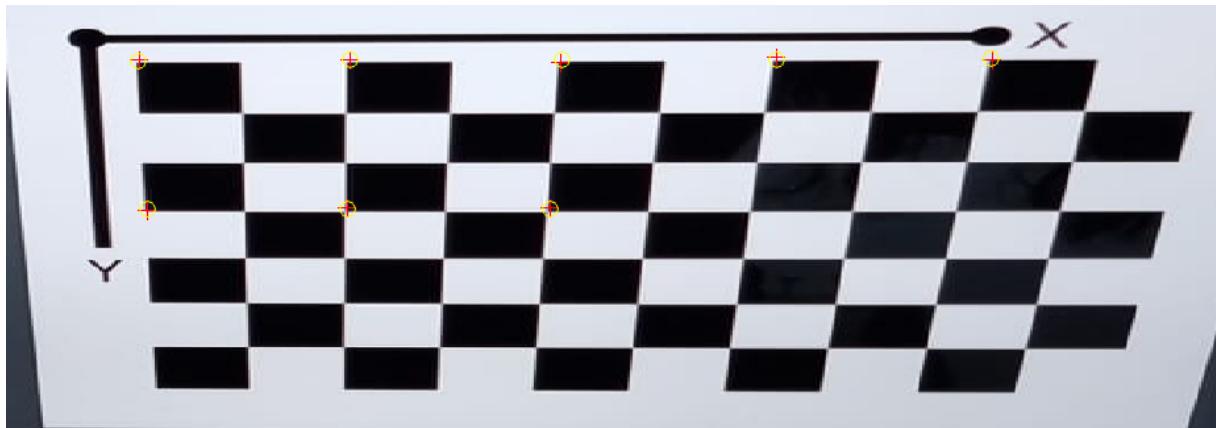
Résultat de notre projection :

```
ReprojPoint =
1.0e+02 *
Columns 1 through 5
6.236112758183700 6.242606862960515 6.248987958635767 6.255258969606961 6.261422720232900
3.502280314497463 3.958267469173141 4.406319627349016 4.846642127211317 5.279433282673833
0.0100000000000000 0.0100000000000000 0.0100000000000000 0.0100000000000000 0.0100000000000000
```

### Méthode 2 :

On a choisi 8 points 3D (dont on a calculé les coordonnées dans le repère du monde, il s'agit des coins supérieurs gauche de quelques carrés de l'échiquier. La taille d'un carré est de 17 mm et l'origine du repère monde est sur le premier point en haut à gauche), puis on fait leurs projections sur une image avec notre matrice de projection et d'un autre coté on sélectionne à la main les 8 points correspondant qu'on censé trouver dans l'image pour pouvoir comparer chaque paire de point.

- Plus rouge (+) : point image calculé par la projection.
- Cercle jaune (○) : point sélectionné à la main.



**Observation :** On note que les points image calculés à partir de la matrice projection et les vrais point image sont presque les mêmes alors on déduit que notre calibration a bien fonctionné.

## 2) Homographie :

Une homographie est une application projective linéaire qui permet un passage entre deux plans, l'intérêt d'implémenter cette fonction est de passer du plan de la mire (d'un point 3D) vers le plan d'image à travers le repère de la caméra (à un point image), cette fonction homographique  $H$  est une restriction de la matrice de projection avec deux vecteurs de rotations seulement (2D).

Cette fonction est utilisée lors du calcul de la pose de la caméra. On a besoin d'au minimum 4 paires de points, on lui fournit :

- Les 4 points images  $m_1$
- Les 4 points dans le plan de la mire  $m_2$

Chaque paire de point génère deux équations linéairement indépendantes, donc on aura 8 équations et comme la matrice  $H$  de taille  $3 \times 3$  est à facteur près on fixe  $H(3,3) = 1$  par exemple, et on calcule les coefficients de  $H$ .

On a utilisé la SVD pour résoudre ce système où la solution est la dernière colonne de la matrice  $V$ , et on retrouve notre Matrice homographie.

### Test de la fonction DLT :

Cette fonction a été tester lors du TP avec les données de l'enseignant.

```
h = zeros(3,3)
m1= [1 2 1; 2 3 1; 1 1 1; 2 4 1]
m2= [1 5 1; 2 1 1; 5 4 1; 8 4 1]

H= myDLT(m1,m2)

h(1,1:3) = H(1:3)
h(2,1:3) = H(4:6)
h(3,1:3) = H(7:9)
format short

m1_h = h*[1 5 1] '
m1_h_n = [m1_h(1)/m1_h(3) m1_h(2)/m1_h(3) 1]

m2_h = inv(h)* [1 2 1] '
m2_h_n = [m2_h(1)/m2_h(3) m2_h(2)/m2_h(3) 1]
```

Les résultats du test sont :

```
m1_h =
0.0621
0.1242
0.0621
```

```
m1_h_n =|
1.0000    2.0000    1.0000
```

```
m2_h =
16.1050
80.5251
16.1050
```

```
m2_h_n =
1.0000    5.0000    1.0000
```

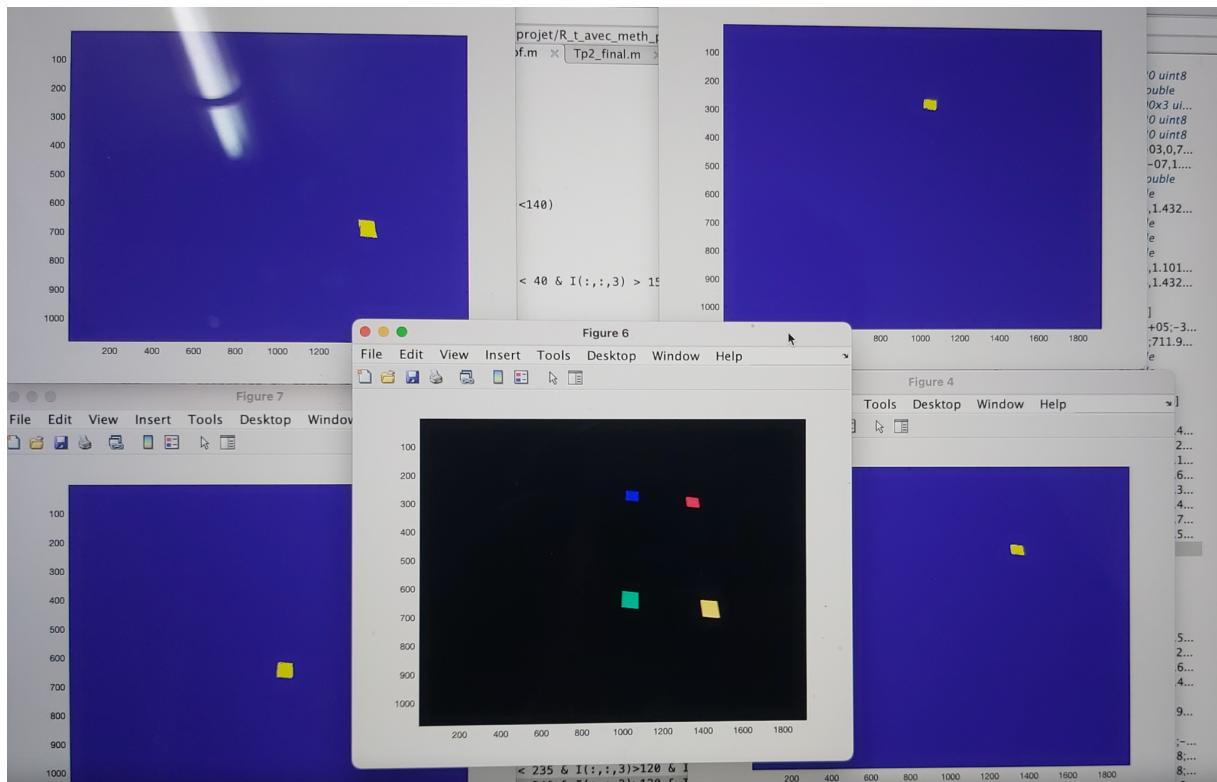
On remarque que le point m1\_h\_n calculé avec l'homographie à partir du point [1 5 1] est bien égale à son point image correspondant dans l'image 2 qui est [1 2 1] et la même chose pour le deuxième point.

### 3) Le Tracking de primitives :

Pour générer l'objet 3D, il faut suivre le mouvement de 4 points bien choisis pour pouvoir lier l'objet à ces derniers et ceci dans chaque photo.

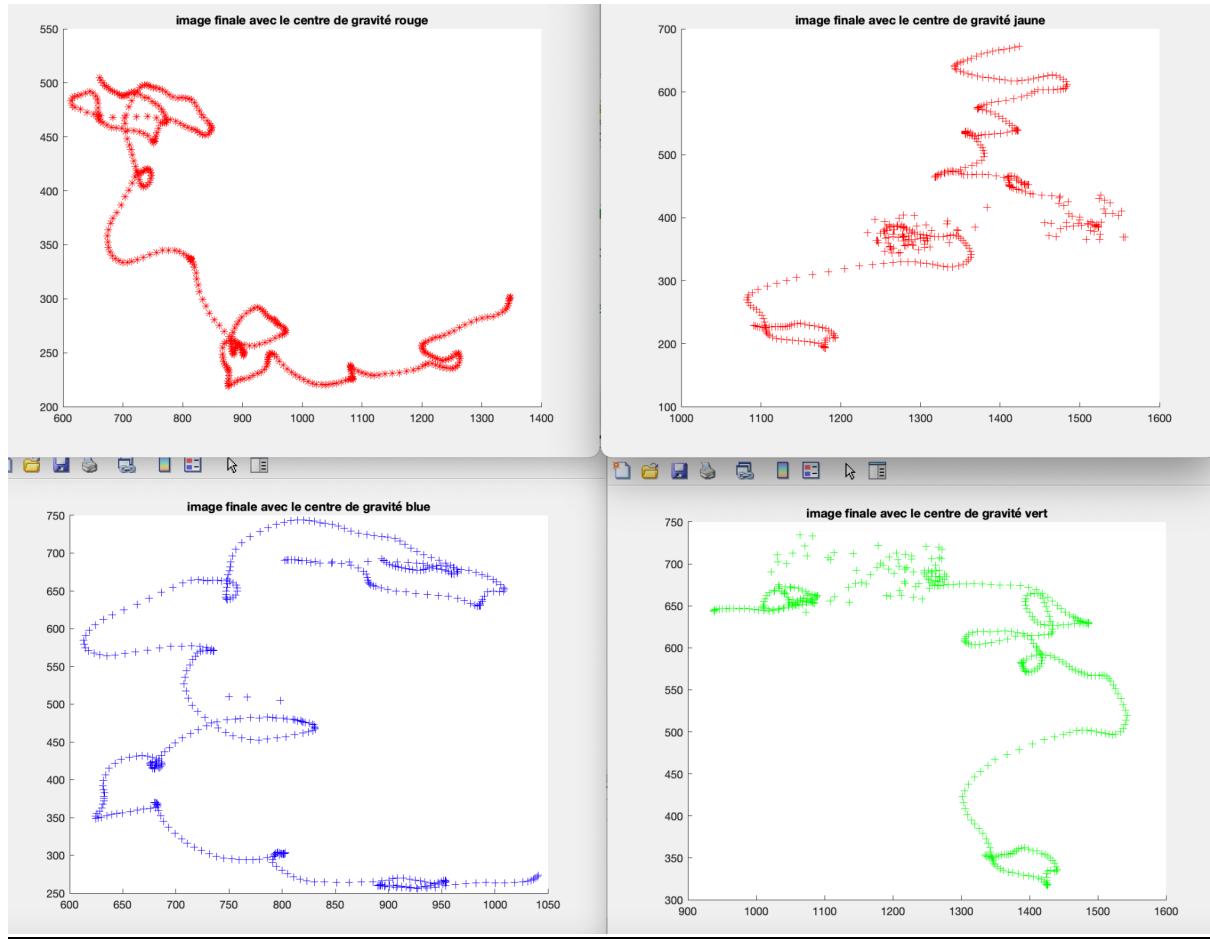
On a filmé une petite vidéo d'un fond noir sur une tablette contenant 4 carrés ( rectangles) de couleurs différentes ( Rouge : r, Vert : g, Jaune : y, Bleu : b), pour calculer les homographies avec lesquelles on déduit les matrices de projection, on a fait le suivi de trajectoire des centres de gravité de chacun des carrés, pour cela on a appliqué des filtres pour extraire chaque carré tout seul et calculer son centre de gravité, qu'on stocke dans des vecteurs qui vont contenir à la fin, la suite des positions des centre de gravité.

On a utilisé quelques connaissances acquises en traitement d'images lors de la L3 (dont l'étiquetage ) pour calculer les centres de gravité, ci-dessous le résultat de l'étiquetage des carrés isolés :



À la fin de l'exécution du programme, on a 4 vecteurs contenant les trajectoires des 4 carrés au long de toute la vidéo (dans toutes les images qui la constituent).

Voici les trajectoires obtenues pour les centres de gravité des 4 carré de couleur différentes :



#### 4) Estimation de la pose :

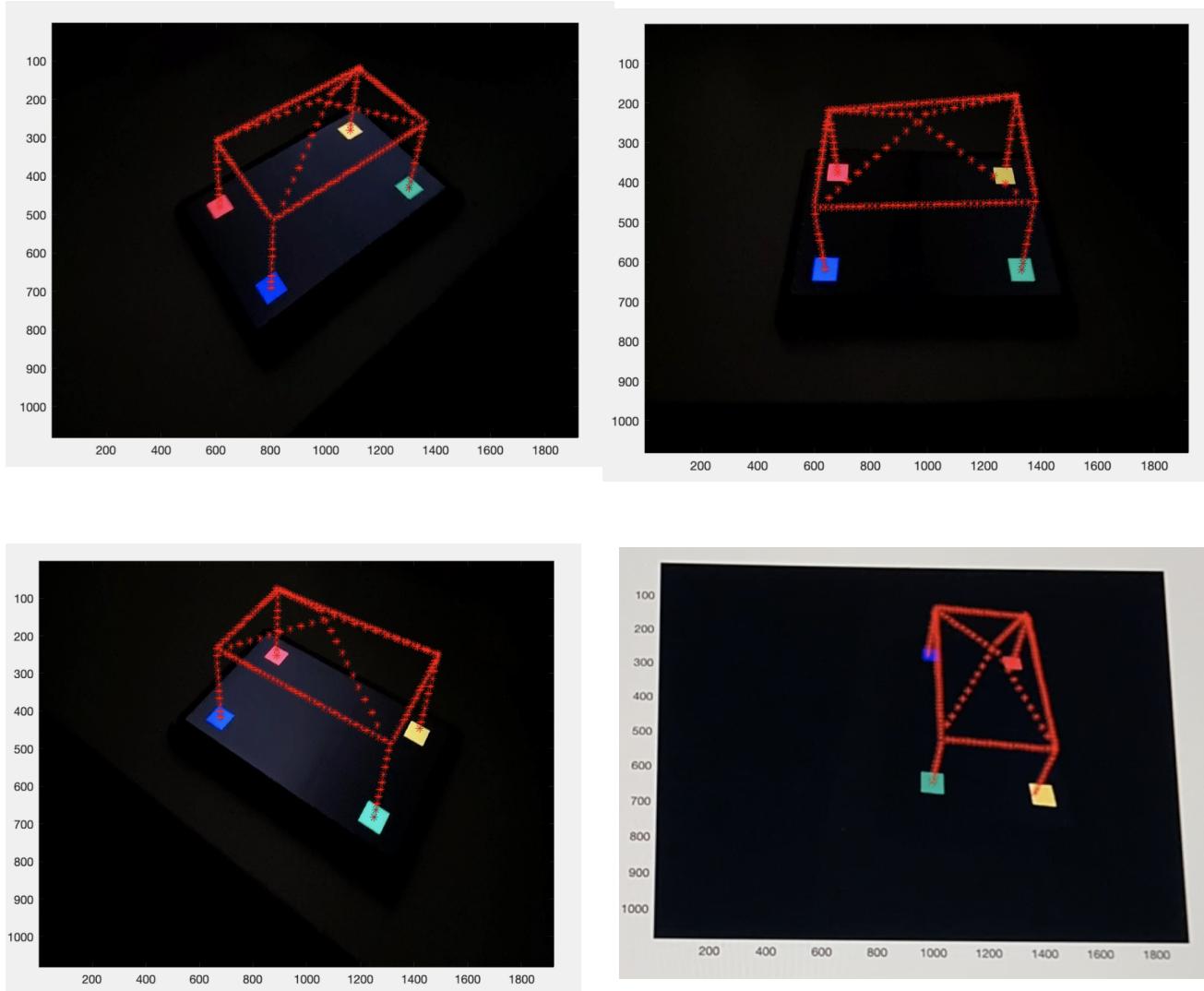
On a utilisé les valeurs contenues dans les vecteurs de suivi de trajectoire pour calculer les homographies entre le plan de la mire et les plans images, pour chaque image, on a les points dans le plan de la mire qui correspondent aux centres de gravités des carrés et on a les 4 positions de ces derniers dans l'image, on utilise la fonction DLT pour calculer l'homographie.

Une fois l'homographie calculée, on calcule la pose de la caméra pour cette image en utilisant une méthode dérivée de la méthode de Zhang, cette dernière a deux solutions on choisit celle dont la composante z du vecteur de translation est positive et on calcul la matrice de projection vu qu'on a déjà la matrice intrinsèque fournie par Matlab dans cameraParams.

On a stocké toutes les matrices de projection dans une matrice à 3 dimensions PP, pour ne pas trop tarder sur leurs calculs lors de la génération de l'objet 3D.

## 5) Génération d'une petite maison :

Pour ce faire, on a réalisé une interpolation dans le repère du monde des lignes qui composent la maison avec de simples fonctions affines. La maison est constituée de 4 lignes verticales qui représentent les piliers et d'autre lignes pour faire le toit de la maison. Par la suite, on a calculé les points images qui forment ces lignes et on les a dessinés sur chaque image.

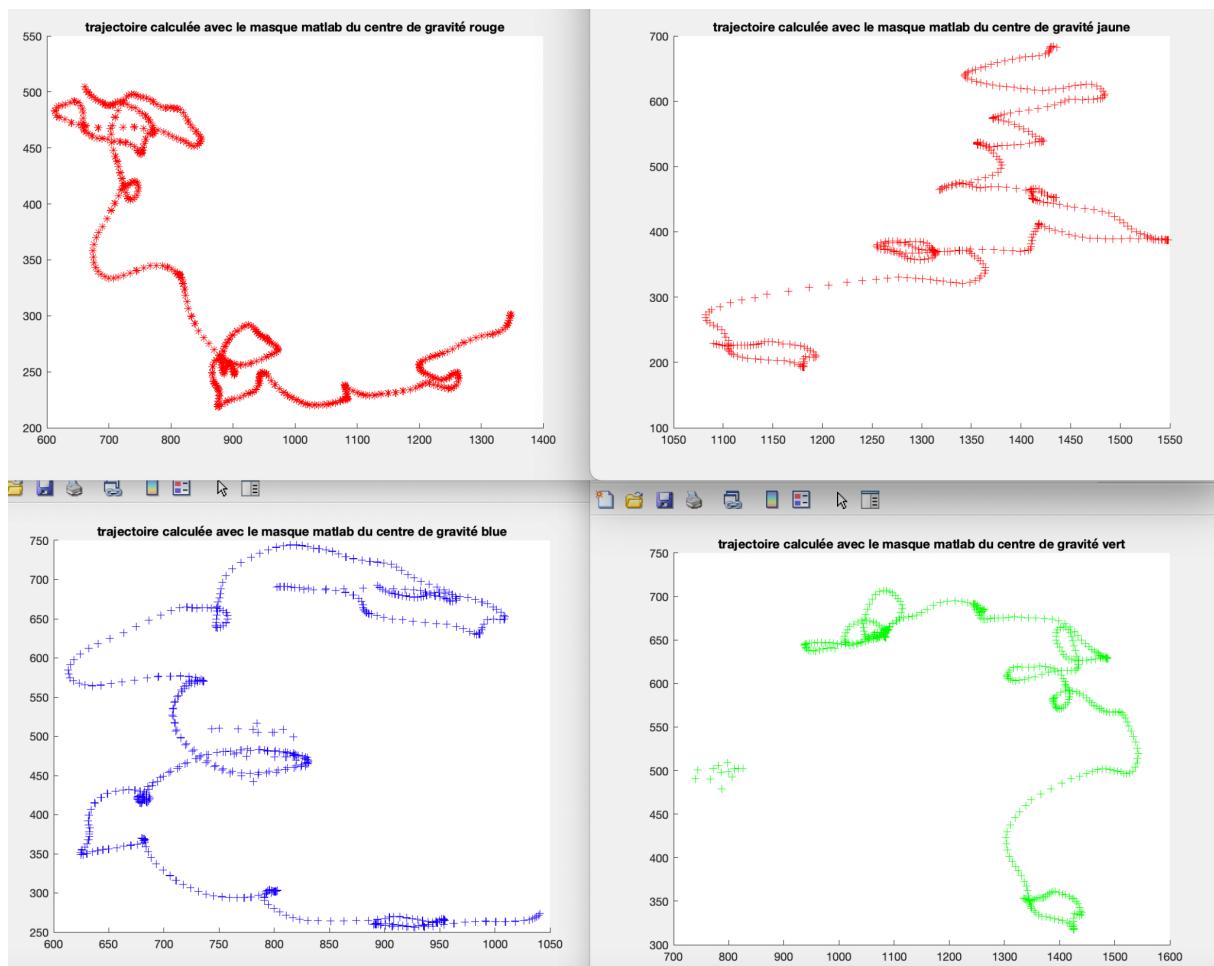


Ceci va nous permettre de générer une suite de figure avec la petite maison dessinée dedans, qu'on transforme en vidéo par la suite.

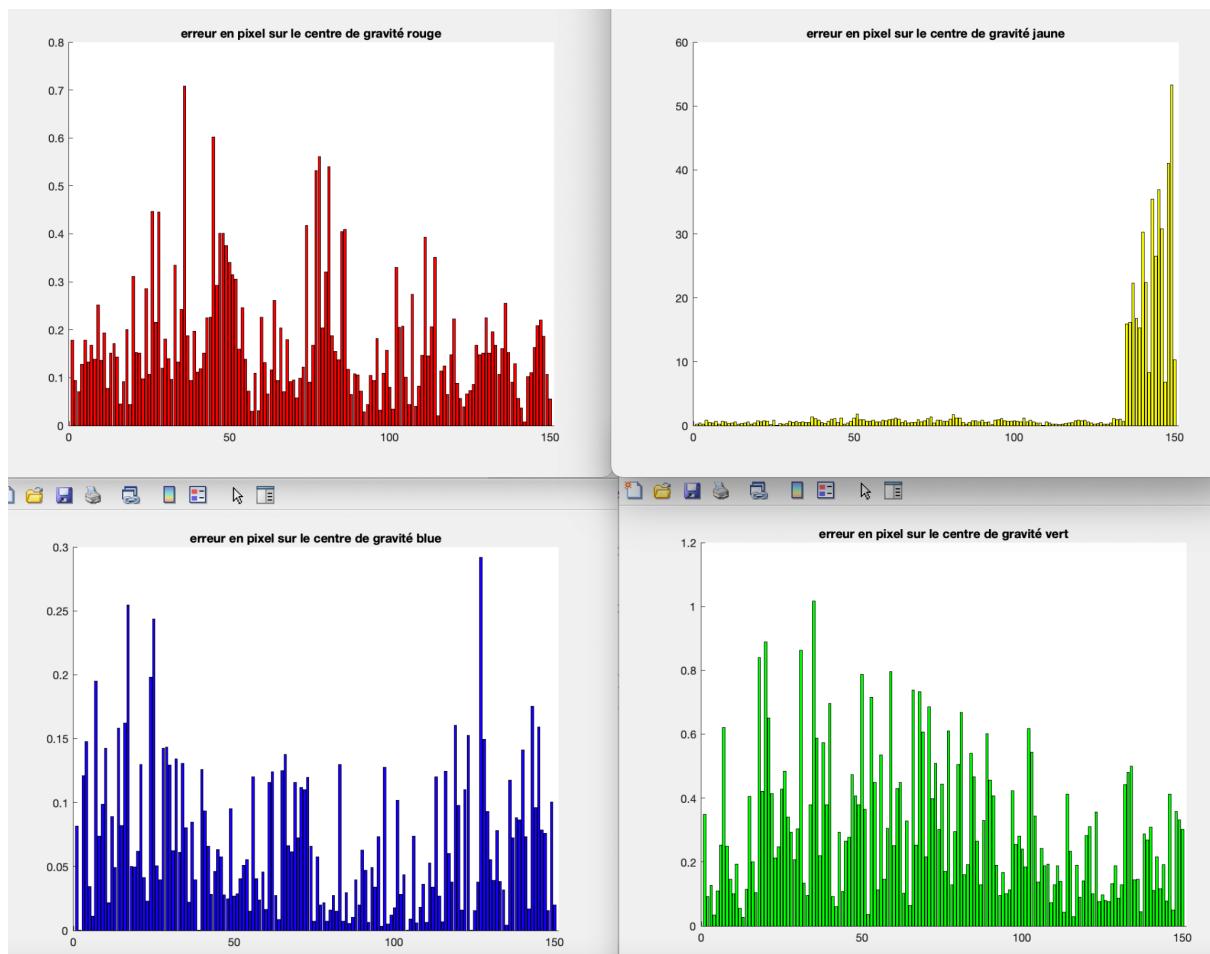
## 6) Évaluation et erreurs de Tracking des centres de gravité :

Pour calculer l'erreur, on a fait le tracking des centres de gravité avec une autre méthode en utilisant les Apps (fonctions intégrées) de Matlab, pour ceci on a créé 4 masques avec la fonction colorthreshold pour extraire les différents carrés pour calculer leurs centres de gravité.

On pour ces nouveaux filtres a trouvé les trajectoires suivantes :



On a calculé l'erreur de suivi de trajectoire :



**Observation :** on a des erreurs minimes sauf vers la fin de la vidéo sur la détection de la couleur jaune où l'erreur arrive jusqu'à 60 pixels.

## 7) Difficultés rencontrées :

Au début on a utilisé un fond blanc, l'extraction des carrés étais beaucoup plus difficile surtout pour la couleur jaune car elle est proche du blanc et du vert.



Après, on a utilisé un fond noir, la plus grande difficulté rencontrée était d'extraire les carrés jaune et vert de chacune des images car ces deux couleurs sont proches et ont une très grande variation de la teinte RGB sur les différentes images, chose qui n'a pas facilité le calcul des filtres et qui a causé les vibrations de la maison vers la fin de la vidéo.

Après l'extraction des carrés, il fallait calculer les centres de gravités, mais les photos binaires (résultats des masques) avaient plusieurs zones (étiquettes), il fallait choisir une taille minimale en pixel à dépasser pour que la zone soit prise en compte dans le calcul du centre de gravité, les choix étaient difficile pour générer exactement les vecteurs des centres de gravité avec des dimensions qui correspondent exactement au nombre de photos disponibles.

## 8) Conclusion :

- Ce projet nous a permis de mettre en œuvre nos connaissances acquises dans cette UE et on a découvert les problèmes qui apparaissent lors de la recherche de solution aux problèmes de traitement d'images.
- On a vu qu'il existe plusieurs méthodes pour faire ce projet, ce qui veut dire que toute idée est la bienvenue. On n'a pas utilisé les fonctions proposées par Matlab pour apprendre le maximum de connaissance possible.
- On a mis en œuvre les programmes des méthodes qui n'étaient auparavant que théorie et on a vérifié l'efficacité de ces derniers.
- C'est une bonne expérience en vision par ordinateur, qui peut nous initier à la 3D qui est très avancée notamment dans le monde du cinéma.