Traitement du Signal

Projet Réalité Augmentée

Joseph Beasse

Paul Chaumont

Thomas Chimbault

Tristan Gonçalves

2A - Groupe 1

Novembre 2022

Table des matières

Introduction **3**

Solution théorique **4**
- Détection des coins 4
  - Traduction de la luminance 4
  - Approche de Canny 4
  - Détecteur de Harris 5
  - Détecteur Multi-échelles 6
  - Suivi des points par trajectographie linéaire 7
- Remplacement du contenu 9
  - Détermination de l'homographie 9
  - Prise en compte de la main 11
  - Incrustation d'un modèle 3D 12

Solution pratique **13**
- Détection et suivi des coins 13
- Remplacement de contenu 16
  - Détection de la main 17
- Dessin 18
- Ajout de contenu 3D 19

Résultats obtenus **20**
- Algorithmie et traitement d'image 20

Organisation **25**
- Répartition du travail 25
- Organisation entre binôme 25

Conclusion et perspectives **27**

Annexes **28**

Introduction

Aujourd'hui, le traitement des images s'applique dans de nombreux domaines : sécurité, médecine, aéronautique... En diversifiant les techniques et procédés, l'Humain est désormais capable d'exploiter une image numérique pour en tirer de l'information, l'améliorer ou bien changer complètement sa nature.

Dans le cadre du projet de Traitement d'images encadré par M. Donias, il a été question d'exploiter les divers procédés acquis au long de ce module dans l'objectif de remplacer le contenu précis d'une vidéo et y ajouter un modèle 3D. La vidéo originale montre une feuille bleue disposée sur une table. Sur cette feuille sont placées des structures type LEGO. La feuille et les structures sont ensuite déplacées par la main d'une personne tout en restant dans le champ de la caméra.

De la description de la vidéo originale naissent diverses contraintes qui sont les fils rouges du projet. Le premier consiste en la détection de la feuille le long de la vidéo. Le second suggère de remplacer le contenu de la feuille tout en prenant en compte l'angle d'acquisition et la main de la personne. En effet, la vidéo finale doit conserver la main de la personne à l'identique. Enfin, le troisième vise à intégrer un modèle 3D en "fils de fer".
Une vidéo étant une succession d'images, le traitement se fera sur chaque image de la vidéo.

Solution théorique

Détection des coins

L'objectif de cette partie est de récupérer les positions, pour chaque image (frame) de la vidéo, des points importants de la feuille afin d'y incruster une image. Parmi ces derniers, on compte les 4 coins, qui délimitent la totalité de la surface de cette dernière (rectangulaire), ainsi que des points des structures bleues et blanches en Lego facilitant la modélisation en 3D (ajout de la hauteur à la surface existante). La position des coins est initialement manuellement au début de la vidéo, l'algorithme a un rôle de suivi de ces coins.

Pour procéder à ce traitement, on va suivre l'approche de Canny (ou détecteur de Canny).
Cette approche indique par le biais d'une valeur numérique forte ou faible si pour chaque pixel de l'image, la position correspond à un coin, une arête ou une zone unie.

Traduction de la luminance

Afin de mettre en place le détecteur, il faut pouvoir mettre en évidence les zones de forts contrastes de Luminance sur l'image. On dispose en effet d'une matrice disposant de 3 informations de couleurs par pixel, le RGB (R = rouge, G = vert et B = bleu).
On souhaite alors créer une nouvelle matrice-image décrivant la luminance à l'aide de la formule suivante:

$$imageLuminance = (0.299*R) + (0.587*G) + (0.114*B)$$

Notre détecteur sera ensuite appliqué sur cette image.

Approche de Canny

Le filtre de Canny est utilisé en traitement d'images pour la détection de contours.
La première étape de ce dernier est un filtrage du bruit de l'image. Il permet d'éliminer les valeurs aberrantes (outliers) qui pourraient induire de faux positifs.
Pour réduire le bruit additif, on choisit d'utiliser un filtre Gaussien.
On calcule alors les Gaussiennes horizontales et verticales (selons les 2 axes du plan):

$$G_a^\sigma(x,y) = -\frac{a}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} avec\ \ a = (x,y)$$

S'ensuit alors le calcul du gradient de l'image, indice de l'intensité des contours.
En effet, Les bords d'une image se situent dans les régions qui présentent de forts changements d'intensité.
Selon Canny, on peut approximer les dérivées d'ordre n par produit de convolution entre la matrice d'intensité (luminance) de l'image et les dérivées d'ordre n d'une gaussienne.
Pour estimer le gradient d'intensité, on convolue alors l'image avec une gaussienne d'ordre 1, normalisée:

$$I_x^{\sigma G} = I * G_x^{\sigma G},\ I_y^{\sigma G} = I * G_y^{\sigma G}$$

σG correspond à un paramètres d'échelle contrôlant la force de la dérivation. Il est égal à 2.
Pour obtenir les contours, on cherche alors les maximums locaux de $\| \nabla I \|$.

Détecteur de Harris

Dans le but de suivre les points de la feuille, il convient d'adopter une méthodologie en cohérence avec cet objectif.
Le détecteur de Harris, développé en 1988, est une méthode populaire de vision par ordinateur utilisée pour repérer les coins dans une image. Elle se base sur une analyse de la covariance locale des gradients de l'image obtenue en récupérant les gradients calculés avec l'approche de Canny. Cela se traduit mathématiquement par la formule suivante :

$$C(x,y) = \begin{bmatrix} \sum_{(u,v)} I_x(x+u,y+v) I_x(x+u,y+v) & \dots \\ \dots & \dots \end{bmatrix}$$

Après ce calcul vient la détermination de l'opérateur permettant de réellement détecter les coins dans une image. Mathématiquement, cet outil s'exprime par la relation :

$$D(x,y) = Det\ (C(x,y)) - \lambda\, Trace(C(x,y))^2$$

pondérateur ~ 0,05

Ainsi, les valeurs de D sont positives au voisinage d'un coin, négatives au voisinage d'un contour et très faibles dans une zone d'intensité constante.

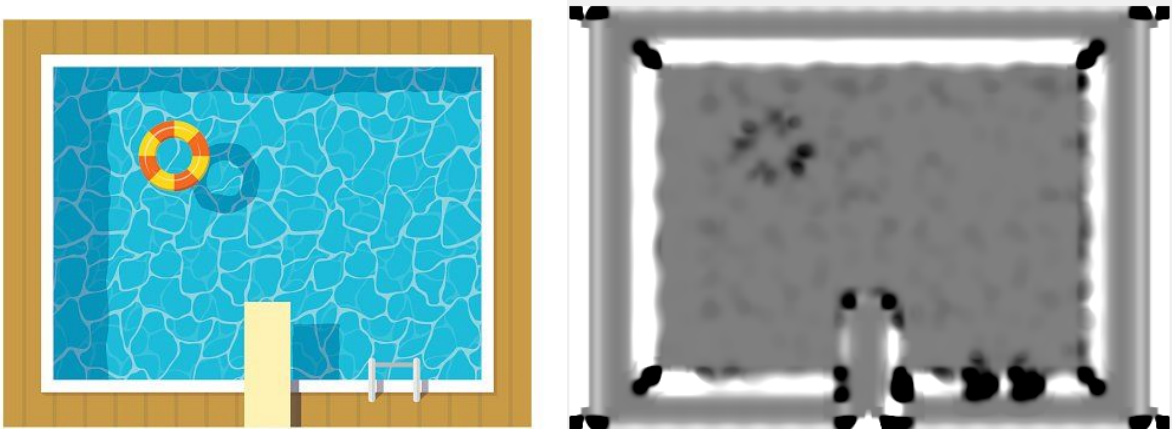


Figure 1. Illustration du détecteur de Harris sur une image de piscine.

En appliquant le détecteur de Harris à une image de la vidéo puis en lui appliquant un colormap "gray" et en inversant les couleurs, on obtient la figure 1. Les coins sont en noir, les contours en blanc et les zones uniformes en gris. On peut tout de même noter quelques imperfections.

Détecteur Multi-échelles

Le détecteur Multi-échelles permet une implémentation matricielle du détecteur de Harris afin de factoriser le code, d'améliorer sa lisibilité, sa complexité temporelle mais surtout d'augmenter ses performances. En effet, certains coins des légos sont difficilement repérables car il y a peu de contrastes entres les points environnants. Pour améliorer sa précision, le détecteur multi-échelles semble plus adapté.

Il combine deux détecteurs de Harris avec pour chacun, un facteur d'échelle σC différent.

On note D1 et D2 deux détecteurs de Harris utilisant respectivement les facteurs d'échelles suivants: $\sigma C1 = 3$ et $\sigma C2 = 5$.

$$D = \min\left(D_1^{\sigma_G,\sigma_{C_1}} \left|D_2^{\sigma_G,\sigma_{C_2}}\right|, \left|D_1^{\sigma_G,\sigma_{C_1}}\right| D_2^{\sigma_G,\sigma_{C_2}}\right)$$

En appliquant la formule ci-dessus, on constate une meilleure netteté pour ce multi-échelle par rapport au simple détecteur de Harris.

Suivi des points par trajectographie linéaire

La trajectographie linéaire des points consiste à trouver le pixel correspondant au coin dans une fenêtre carrée de taille donnée (coté = 37 pixels dans ce projet).
Tout d'abord, la position du coin est initialisée par un clic de souris.

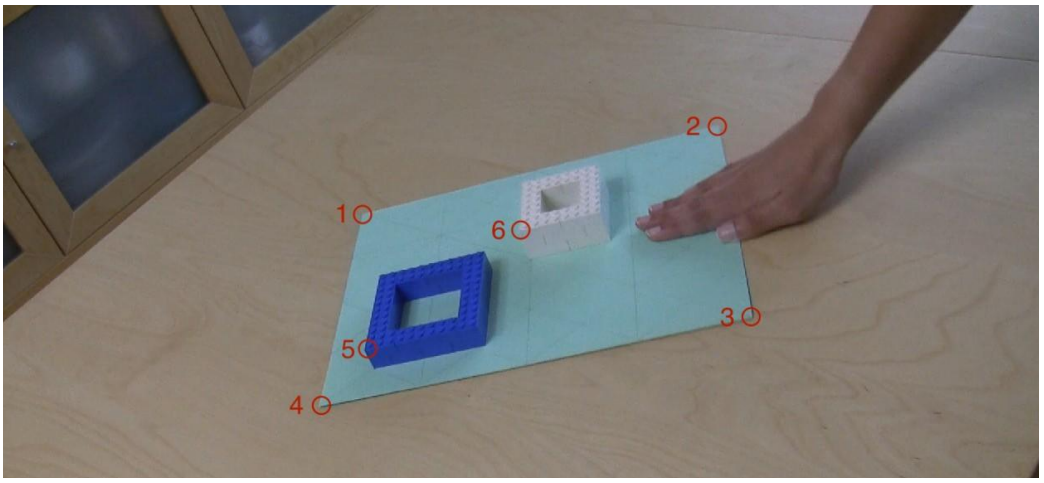


Figure 2. Illustration montrant l'initialisation de 6 points

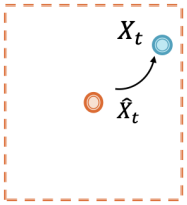
Ensuite, on recherche le maximum local dans la fenêtre de taille donnée autour de ce point initial. Ce dernier devient alors un coin. On souhaite suivre son évolution tout au long de la vidéo. Pour cela, on prédit linéairement la position du centre de la fenêtre de recherche du coin sur l'image suivante, puis on récupère le maximum local qui devient la position du coin. On répète cette méthode pour toutes les prochaines frames de la vidéo.
La fonction de prédiction linéaire est la suivante:

$$\hat{X}_t = X_{t-1} + \frac{X_{t-1} - X_{t-2}}{2}$$

$\hat{X}_t$ = Position du centre de la fenêtre de recherche de l'image t

X_{t-1} = Position du coin à l'image t-1

X_{t-2} = Position du coin à l'image t-2



Chaque position de coin est le résultat de la recherche d'un maximum local autour de la position prédite linéairement sur une fenêtre de taille donnée. (voir schéma à gauche).

Remplacement du contenu

Le but de cette partie était de remplacer la feuille mobile de la vidéo par une autre image que nous avions choisie (un sol lunaire), et de garder cette image sur la feuille tout au long de la vidéo. Le but est donc de pouvoir relier des positions de pixel de l'image de base à des pixels sur la vidéo et cela, pixel par pixel. Notons P2=[x2,y2] les coordonnées d'un point dans une frame de la vidéo et P1=[x1,y1] les coordonnées du point équivalent dans l'image. La plupart des transformations s'écrivent sous la forme:

$$P2 = A * P1 + B$$

Pour des soucis de simplicité nous ajoutons une dimension au vecteur des positions, on obtient des coordonnées homogènes. L'équation s'écrit alors:

$$P2 = H * P1$$

avec:

$$\overline{P2} = [x2, y2, s]$$

$$\overline{P1} = [x1, y1, 1]$$

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix}$$

Détermination de l'homographie

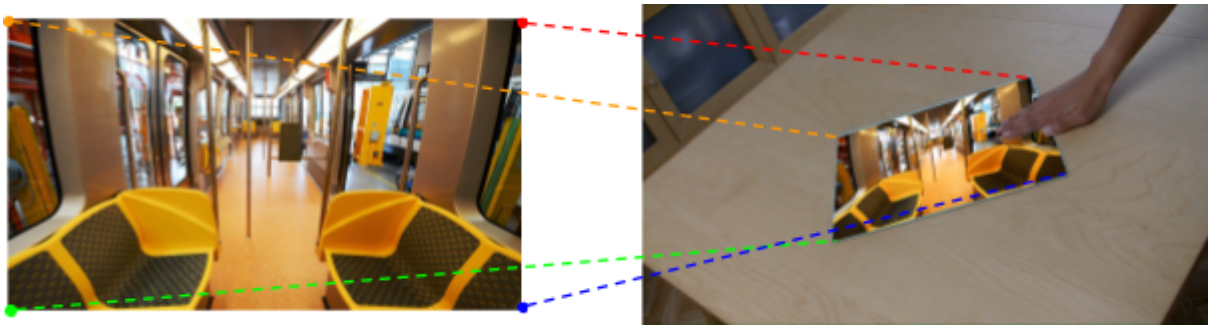


Figure 3. Illustration d'homographie de l'image sur la vidéo

L'objectif est donc de trouver une matrice de projection homographique permettant l'incrustation d'une image "dans" la feuille bleue se trouvant dans la vidéo. Il est intéressant de noter que le coefficient H_{33} est le coefficient d'agrandissement, il vaudra toujours 1 dans notre cas.

La matrice homographique permet de convertir des coordonnées de l'image initiale en coordonnées sur la vidéo. Nous pouvons obtenir l'homographie grâce à la connaissance de points particuliers équivalents entre l'image initiale et la vidéo.

Grâce à 2 points équivalents nous pouvons obtenir les équations suivantes:

$$x_2 = \frac{H_{11}x_1 + H_{12}y_1 + H_{13}}{H_{31}x_1 + H_{32}y_1 + 1}$$

$$y_2 = \frac{H_{21}x_1 + H_{22}y_1 + H_{23}}{H_{31}x_1 + H_{32}y_1 + 1}$$

Qui est équivalent à :

$$x2 = H_{11}x_1 + H_{12}y_1 + H_{13} + 0 + 0 + 0 - x_1x_2H_{31} - y_1x_2H_{32}$$

$$y2 = 0 + 0 + 0 + H_{21}x_1 + H_{22}y_1 + H_{23} - x_1y_2H_{31} - y_1y_2H_{32}$$

Grâce à ces 2 équations nous en obtenons une nouvelle:

$$AH = B$$

avec :

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_2 & -y_1x_2 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_2 & -y_1y_2 \end{bmatrix}$$

$$H = [H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}]^t$$

$$B = [x_2, y_2]^t$$

H a 8 inconnus, il suffit donc d'avoir 4 fois 2 points équivalents pour pouvoir déterminer H. On peut agrandir A et B en connaissant plusieurs points. D'ailleurs avoir plus que 4 fois 2 points équivalents permet de déterminer plus précisément H.

Figure 4. Illustration d'homographie de l'image sur la vidéo

Comme dit précédemment, l'application de la matrice d'homographie à l'image permet donc d'associer les pixels d'une image 2D à la position qu'ils auraient eu si la feuille avait été "physiquement" remplacée par cette image. Pour se calibrer sur la vidéo, les 4 coins de la feuille ont été sélectionnés et indiqués au préalable.

Prise en compte de la main

Afin de détecter la main, il a tout d'abord fallu déterminer une zone rectangulaire dans laquelle la main se situe et appliquer la matrice d'homographie sur celle-ci, afin d'obtenir les coordonnées relatives sur la feuille.

Ensuite, étant donné le contraste entre la couleur de la main et la feuille de la vidéo, il a été décidé de se baser sur la teinte (*Hue*) en utilisant la formule disponible pour la conversion de RGB vers HSL, à savoir :

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G'-B'}{\Delta} mod 6\right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , C_{max} = B' \end{cases}$$

avec

$R' = R/255$; $G' = G/255$; $B' = B/255$

$Cmax = \max(R', G', B')$; $Cmin = \min(R', G', B')$; $\Delta = Cmax - Cmin$

Ensuite, la couleur de peau de la main a été sélectionnée grâce à un outil "pipette de couleur". À partir de ce RGB, un delta de manœuvre a été sélectionné, pour prendre en compte les variations de teintes de la main (ongles, imperfections, etc) et la conversion a été effectuée.

Cela a permis de pouvoir détecter les pixels de la main dans la zone préalablement désignée, puis lors de l'incrustation d'image, d'éviter l'occultation de la main.

Incrustation d'un modèle 3D

Après avoir changé la feuille bleue par une autre image sans effacer les pixels de la main, nous avions comme but d'incruster un modèle 3D dans la vidéo.
Tout d'abord il a fallu créer la fonction permettant de dessiner sur une image un trait allant d'un point A à un point B. Pour cela nous avons fait une interpolation linéaire entre le point A et le point B L fois.

$$L = E[\alpha\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}]$$

avec α le paramètre de suréchantillonnage et E la partie entière

Cependant de cette manière le trait est trop fin. Pour grossir le trait nous pouvons appliquer plusieurs fois cette fonction avec les points A et B décalés selon la normal au trait initial.

Ensuite il nous a fallu déterminer la matrice de passage de la 3D vers la 2D. Pour cela nous avons procédé de la même manière que l'homographie, à des points de la vidéo nous associons des coordonnées en 3D. Cette fois-ci il nous faut 6 points équivalents pour déterminer la matrice. En effet, les valeurs de la matrice P de passage de la 3D vers la 2D suit ces équations qui a 11 inconnus:

$$AP = B$$

avec :

$$A = \begin{bmatrix} x_1 & y_1 & z_1 & 1 & 0 & 0 & 0 & 0 & -x_1x_2 & -y_1x_2 & -z_1x_2 \\ 0 & 0 & 0 & 0 & x_1 & y_1 & z_1 & 1 & -x_1y_2 & -y_1y_2 & -z_1y_2 \end{bmatrix}$$

$$P = [P_{11}, P_{12}, P_{13}, P_{14}, P_{21}, P_{22}, P_{23}, P_{24}, P_{31}, P_{32}, P_{33}]^t$$

$$B = [x_2, y_2]^t$$

Enfin il a fallu choisir quel modèle 3D nous allions utiliser. Nous avons décidé d'utiliser un fichier stl représentant une soucoupe volante. Les fichiers stl ayant pour but final de permettre le rendu 3D d'objet en utilisant des triangles, ces derniers ont la structure suivante:

- un tableau de points avec leurs coordonnées
- un tableau de triangles représentés par les indices des 3 points formant le triangle

Solution pratique

Pour implémenter les algorithmes décrits précédemment, nous avons utilisé le langage de programmation MATLAB. L'environnement de développement MatLab permet de manipuler efficacement les tableaux à plusieurs dimensions, ce qui en fait un outil très utilisé pour le traitement d'images. Certaines spécificités de ce langage doivent être prises en compte. Par exemple, il faut faire attention à bien différencier les cas où nous traitons des coordonnées matricielles des cas où nous traitons des coordonnées "image". Les coordonnées X et Y sont inversées entre les deux cas. Voici les différentes fonctions que nous avons mises en place pour réaliser le projet.

Détection et suivi des coins

Dans le but de suivre les coins désirés, une fonction nommée Initialisation a été implémentée. Celle-ci est utilisée pour récupérer les coordonnées de tous les points sélectionnés dans le cadre du suivi.

Initialisation function			
Description	***Paramètres d'entrée***		***Sortie***
Récupération des points sélectionnés	coordonnesGInput	Matrice composée des coordonnées en x et y de tous les points sélectionnés	coordInit : vecteur composé des coordonnées en x et y de tous les points sélectionnés
	n	Nombre de points à sélectionner	

Une fois la récupération des points et de leurs coordonnées effectuées, il convient de repérer ces points à chaque frame de la vidéo. Pour ce faire, une fonction nommée Harris a été créée.

Harris function			
Description	***Descriptif des paramètres d'entrée***		***Sortie***
Fonction implémentant le détecteur de Harris.	sigmaG	Paramètres du détecteur de Harris multi-échelle	D : points d'intérêts de l'image
	sigmaC1		
	sigmaC2		
	image	Frame de la vidéo	

La fonction CreateList utilise une liste de positions prédites pour l'ensemble des points et implémente une nouvelle liste de taille 2*n (abscisses et ordonnées pour les n points) des positions réels des coins (maximum locaux sur une fenêtre de taille W=37).

CreateList function			
Description	***Paramètres d'entrée***		***Sortie***
Fonction implémentant la liste des positions des coins suivis par frame	coordonnées	liste des positions prédites linéairement	maxList: liste des coordonnées réelles des coins
	thImage	image en cours de traitement filtré par Harris	
	n	nombre de points suivis	

On a mentionné précédemment la prédiction linéaire des points, la fonction CoordMaxCalcul est étroitement liée avec CreateList, en effet, pour chaque point prédit, cette nouvelle fonction observe son entourage (carré de côté W=37 pixels) et récupère la valeur maximale.

CoordMaxCalcul function			
Description	***Paramètres d'entrée***		***Sortie***
Calcul de maximum d'intensité locale sur une fenêtre de taille déterminée => Identification d'un coin	coordonnées	liste des positions prédites linéairement	coordMax: liste des positions (X,Y) pour chaque point suivi
	thImage	image en cours de traitement filtré par Harris	

La fonction RecupCoins englobe toutes les autres méthodes citées précédemment. Elle implémente le suivi des points sur toutes les frames de la vidéo et permet d'obtenir une matrice comprenant toutes ces dernières. Elle définit également la prédiction linéaire qui a un aspect statique lors de la phase d'initialisation puis dynamique ensuite.

<table>
<tr><th colspan="4">RecupCoins
function</th></tr>
<tr><th>Description</th><th colspan="2">Paramètres d'entrée</th><th>Sortie</th></tr>
<tr><td rowspan="6">Implémente le détecteur de Harris sur chaque image de la vidéo puis récupère les points voulus à l'aide d'une prédiction linéaire suivi d'une recherche de maximums locaux sur une fenêtre</td><td>vidObj</td><td>vidéo définie comme un "VideoReader"</td><td rowspan="6">coord: matrice de 325 lignes de 2*n coordonnées (X,Y) des points suivi pour chaque frame de la vidéo</td></tr>
<tr><td>sigmaG</td><td rowspan="3">Paramètres du détecteur de Harris multi-échelle</td></tr>
<tr><td>sigmaC1</td></tr>
<tr><td>sigmaC2</td></tr>
<tr><td>n</td><td>nombre de points suivis</td></tr>
<tr><td>coordInit</td><td>coordonnées initiales des points à suivre (donnés manuellement)</td></tr>
</table>

Remplacement de contenu

Pour le remplacement de contenu, la tâche s'est axée sur 3 parties majeures :

- La création de la matrice d'homographie et son application
- La prise en compte de la main afin de ne pas l'effacer
- L'ajout de l'objet 3D sous forme de "fil de fer"

Pour la première tâche, deux fonctions ont été créées, à savoir **GetH** qui permet de fournir la matrice d'homographie nécessaire, et **ChangeFeuilleBlanche**, qui réunit la majorité des informations que l'on veut traiter (les différentes images + les coordonnées utiles) et qui va amener à l'application de l'homographie.

GetH function			
Description	***Paramètres d'entrée***		***Sortie***
Fonction fournissant la matrice d'homographie	x	Matrice des abscisses des points de référence sur la vidéo	Matrice H
	y	Matrice des ordonnées des points de référence sur la vidéo	
	X	Matrice de largeur de la feuille	
	Y	Matrice de hauteur de la feuille	

ChangeFeuilleBlanche function			
Description	***Paramètres d'entrée***		***Sortie***
	`ancienneImage`	Frame de la vidéo à modifier	
	`imageFeuille`	Image à incruster sur la feuille	
	X	Coordonnées X des points de référence	

Application de l'homographie sur une frame	Y	Coordonnées Y des points de référence	Nouvelle frame sur laquelle l'homographie est appliquée
	XMain	Coordonnées X du rectangle contenant la main	
	YMain	Coordonnées Y du rectangle contenant la main	

Détection de la main

Le processus de sélection de la main se découpe en plusieurs phases : tout d'abord, il faut définir une zone de recherche dans laquelle la main se situe (fonction **SelectCadreMain**) puis, il faut parcourir cette zone en vérifiant si le pixel parcourut est bleu (correspondant à un pixel de la feuille - fonction **EstBleu**) ou non, auquel cas il correspond à la main, et ne doit donc pas être remplacé.

SelectCadreMain function			
Description	***Paramètres d'entrée***		***Sortie***
Définit le cadre dans lequel on doit chercher la main	imageFeuille	Image que l'on veut transférer sur la feuille bleue	**<u>2 matrices :</u>** 1 matrice de coordonnées X 1 matrice de coordonnées Y
	X	Coordonnées X des points de référence sur une frame	
	Y	Coordonnées Y des points de référence sur une frame	

EstBleu function			
Description	***Paramètres d'entrée***		***Sortie***
Vérifie si le pixel sur la vidéo est bleu ou non	Color	RGB (sous forme de matrice [R,G,B]) de la	Booléen

Dessin

Pour le dessin, il a tout d'abord fallu déterminer le nombre de points considérés comme utiles pour effectuer un tracé entre le point de départ *A* et le point d'arrivée *B*. Pour cela, on cherche la partie entière de la distance euclidienne séparant ces points A et B, que l'on multiplie d'un facteur alpha, nommé paramètre de sur-échantillonnage. Ce dernier permet d'améliorer le tracé et de diminuer la présence de "trous" lorsque ce dernier est plus épais.

DrawSeg function			
Description	***Paramètres d'entrée***		***Sortie***
Fonction permettant de tracer un segment entre deux points	x1	Couple de coordonnées (x,y) du point de départ	Image comportant le trait dessiné
	x2	Couple de coordonnées (x,y) du point d'arrivée	
	I	Image sur laquelle on veut dessiner	
	alpha	paramètre de sur-échantillonnage	

DrawLineFat function			
Description	***Paramètres d'entrée***		***Sortie***
Fonction utilisant *DrawSeg* pour tracer un trait plus épais	x1	Couple de coordonnées (x,y) du point de départ	Image comportant le trait épais dessiné
	x2	Couple de coordonnées (x,y) du point d'arrivée	
	I	Image sur laquelle on veut dessiner	
	alpha	paramètre de sur-échantillonnage	

Ajout de contenu 3D

Afin d'ajouter le contenu 3D sur la vidéo, en partant de l'image à changer par homographie, la frame de la vidéo et l'objet 3D en format stl, il faut passer par plusieurs étapes : la décomposition du modèle stl en divers segments, le passage de ces coordonnées en 3D, puis enfin l'intégration du tout dans la frame de la vidéo. Pour cela, nous avons séparé le processus en 3 fonctions principales, chacune ayant son rôle.

ConvertModelToSeg function			
Description	***Paramètres d'entrée***		***Sortie***
Convertit un STL en segments	points	Liste des points contenus dans le stl	Matrice de tous les segments de l'objet 3D
	connectivityList	Liste des vecteurs reliant les points	

GetP function			
Description	***Paramètres d'entrée***		***Sortie***
Fonction fournissant la matrice de passage 2D à 3D	x	Matrice des abscisses des points de référence sur la vidéo	Matrice P
	y	Matrice des ordonnées des points de référence sur la vidéo	
	z	Matrice des points surélevés de référence sur la vidéo	
	X	Matrice de largeur de la feuille	
	Y	Matrice de hauteur de la feuille	

Add3D function			
Description	***Paramètres d'entrée***		***Sortie***
Trace les traits dans l'environnement 3D	anciennelmage	Image que l'on veut modifier	Image comportant le trait dessiné dans les 3 dimensions
	X	Coordonnées en X des points de repère de la dimension	
	Y	Coordonnées en Y des points de repère de la dimension	
	Seg	Matrice de points séparant les segments	

Résultats obtenus

Algorithmie et traitement d'image

Chaque étape de notre projet a amené à des résultats intermédiaires que l'on va développer par la suite. Pour cela, on va réaliser les étapes du traitement successif sur une image de la vidéo. Pour que la main soit bien présente sur cette image, on a sélectionné la frame N°200/365.

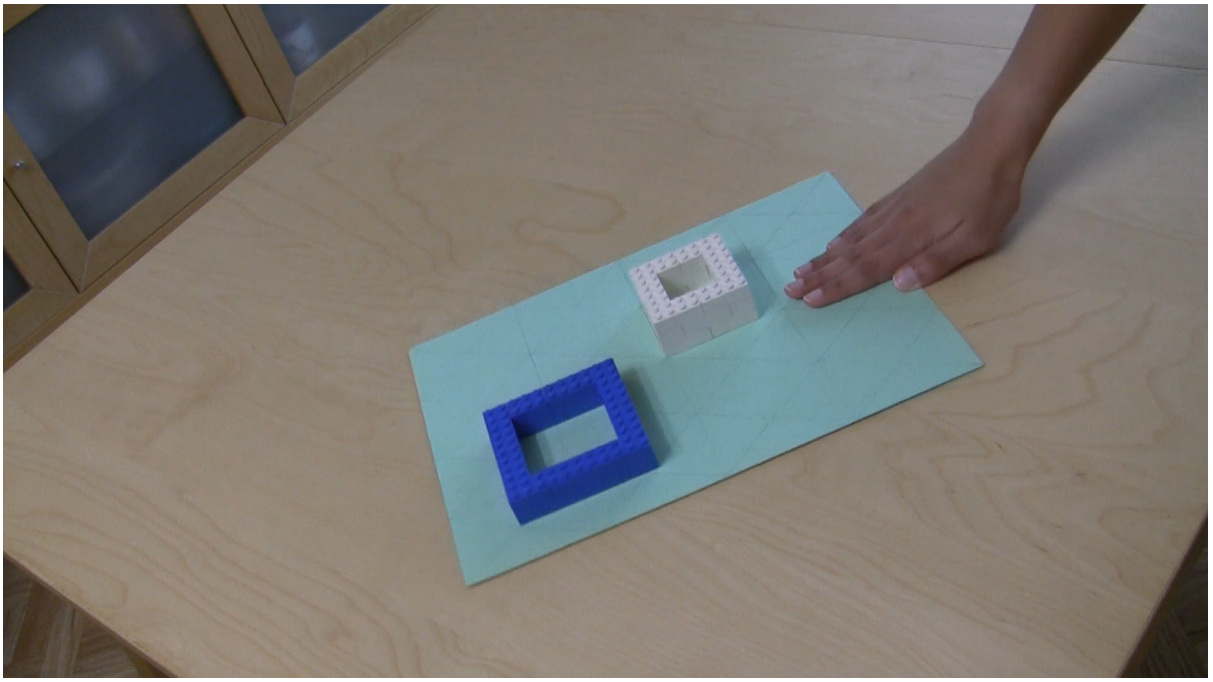


Figure 5. 200^{è} frame de la vidéo.

Dans un premier temps, on effectue la détection des coins sur cette Image en appliquant le détecteur multi-échelle de Harris. On initialise notre recherche pour 6 points particuliers, les 4 coins de feuille et 2 coins des structures bleue et blanche. On applique sur cette image filtré un color map et une inversion de couleurs puis on regarde quels éléments sont représentés en noir, on devrait y retrouver les 6 coins souhaités.

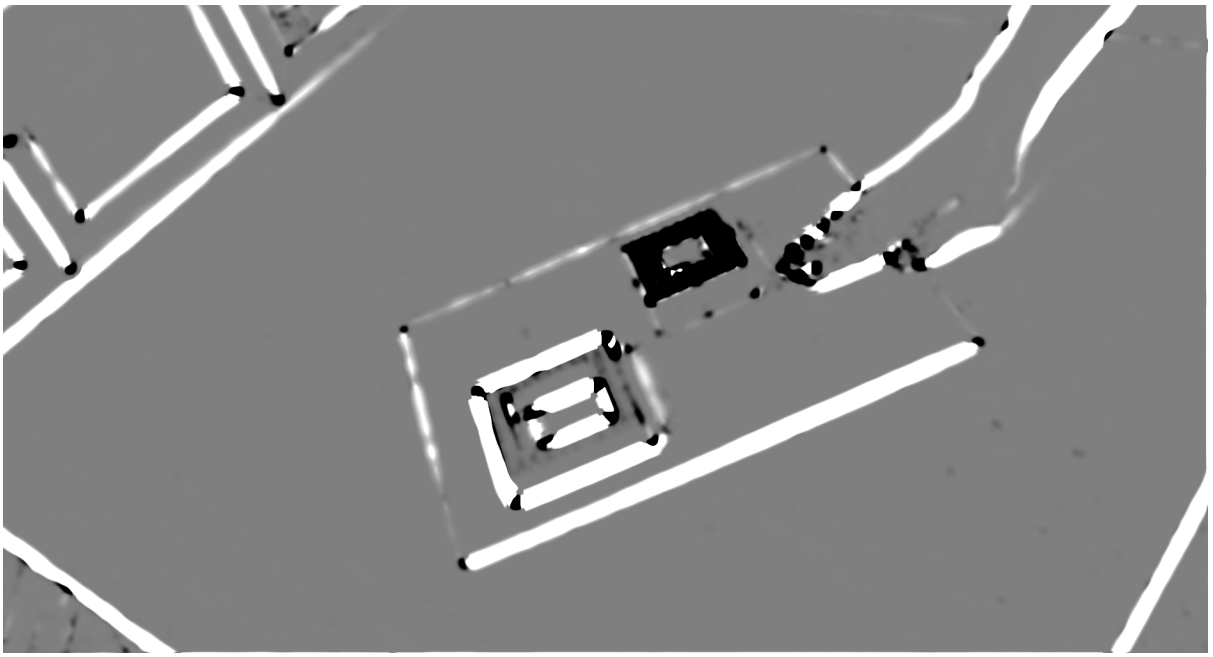


Figure 6. Détecteur de Harris sur la 200^{è} frame de la vidéo avec un "gray" colormap.

Ce résultat est témoin du bon fonctionnement du détecteur de Harris multi-échelle implémenté. Par la suite, il faut prédire linéairement la position des coins entre chaque frame pour accélérer, recentrer et améliorer la recherche des coins. La fonction nous renvoyant la liste des points réels trouvés par maxima locaux est **CreateList ( ).**

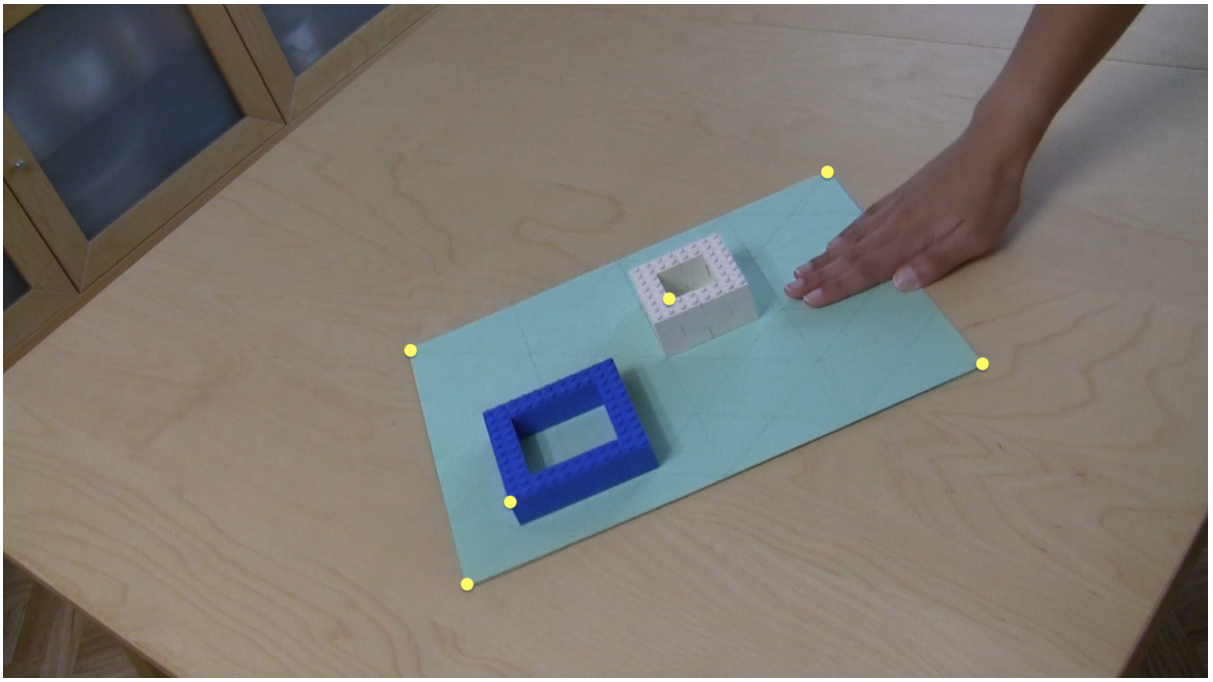


Figure 7. Suivi des points et représentation de leurs coordonnées en jaune sur la 200^{è} frame.

Maintenant que tous les coins nécessaires ont été détectés, ils sont utilisés afin de projeter un contenu sur la feuille. Pour cela, deux étapes sont nécessaires. La première est la récupération de la matrice d'homographie qui permet la déformation de l'image suivante pour la placer dans le repère de la feuille, elle est effectuée par la fonction **GetH ( )**.

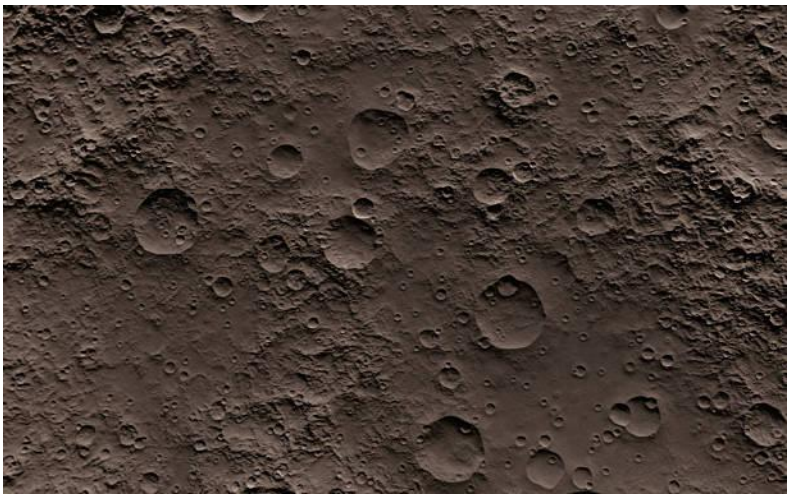


Image 1. Sol lunaire, image à placer sur la feuille.

La deuxième étape consiste à sélectionner la zone où la main est positionnée afin de ne pas l'effacer lors du changement d'image, cette opération est effectuée par la fonction **SelectCadreMain ( ).**

Enfin l'homographie est appliquée à l'image sur la vidéo avec la fonction **ChangeFeuilleBlanche ( )**. pour remplacer et projeter le contenu. On obtient le résultat suivant:

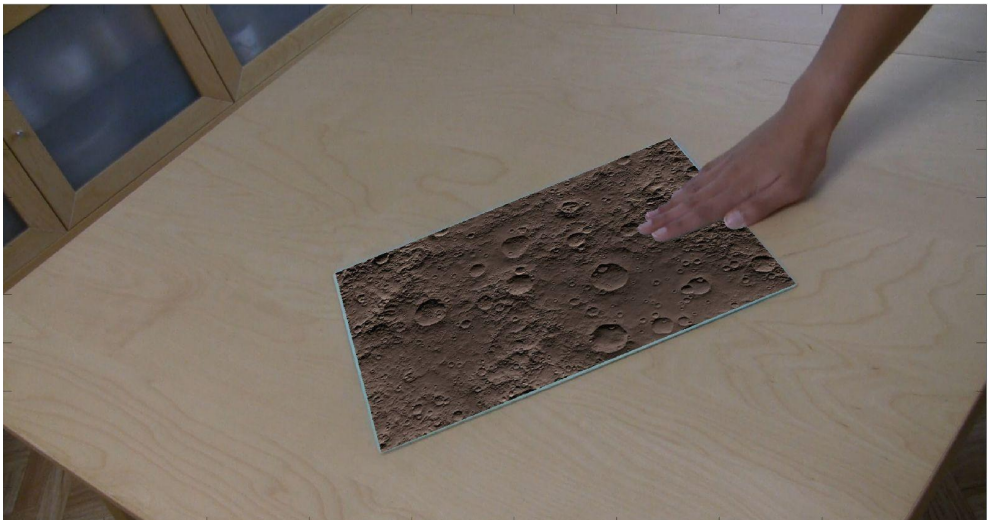


Figure 8. Homographie et remplacement du contenu de la feuille par un sol lunaire.

La dernière étape du projet est le remplacement du contenu 3D. Les coordonnées récupérées sur les 2 blocs de LEGO sont utilisées pour générer une dimension supplémentaire.
Le modèle à ajouter doit être un fichier d'extension .stl, la fonction **ConvertModelToSeg ( ).**
Une fois le modèle passé au format de segments on l'ajoute sur la vidéo dans le bon repère à l'aide de **Add3D ( )**, cette dernière colore les pixels entre chaque segment de l'objet.
Le résultat de la "soucoupe" blanche est donné sur l'image suivante:

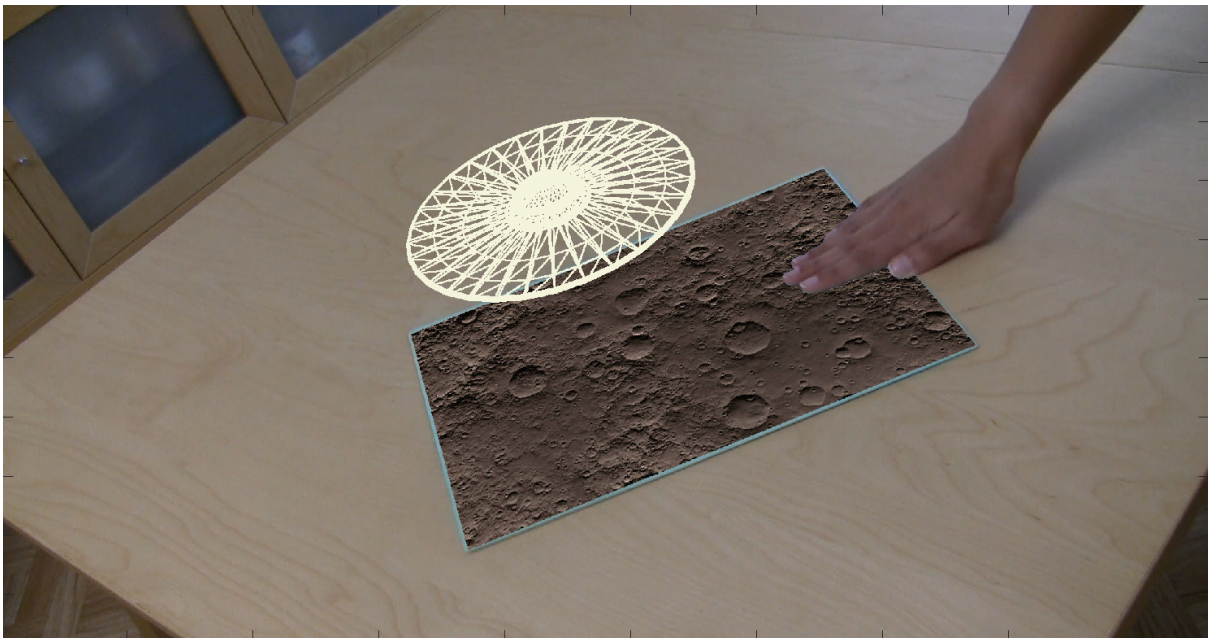


Figure 9. Ajout du modèle 3D "soucoupe volante sur un sol lunaire" sur la frame n°200.

Enfin, ce traitement est effectué pour l'ensemble des 365 frames de la vidéo.

Organisation

Répartition du travail

Ce travail a été réalisé sur une période d'environ un mois. Dès le lancement du projet, il a été décidé de diviser le projet en deux binômes abordant chacun une partie précise. Le premier groupe s'est focalisé sur la détection, le suivi et la récupération des points d'intérêts. Le second s'est intéressé au remplacement de la feuille bleue par un contenu différent en prenant en compte la main. Le travail du second dépendant des résultats du premier, les deux binômes se sont concertés afin de convenir de la forme des résultats échangés.

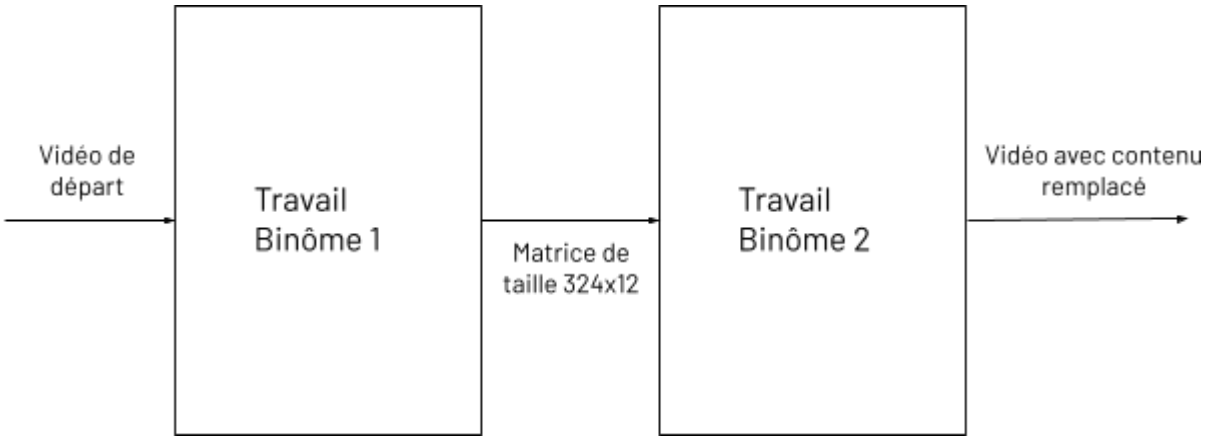


Figure 9. Organisation sur la structure des données échangées

Enfin, une fois le travail des deux binômes achevé la dernière partie a été d'ajouter un modèle 3D au contenu de la vidéo. Pour ce faire, le premier binôme s'est occupé de déterminer une transformation de la 3D vers la 2D. Le second groupe s'est chargé d'ajouter la structure 3D en traçant les segments composant la structure en "fils de fer".

Organisation entre binôme

Tout au long du projet, chaque binôme a avancé de son côté en organisant régulièrement des réunions de travail. Les séances de TP encadrées par M. Marc Donias ont notamment servi à échanger avec l'autre binôme sur l'avancement général, les difficultés rencontrées et sur les potentiels changements dans la structure des données attendues.

Conclusion et perspectives

Ce projet de vision par ordinateur a ainsi permis de réfléchir sur des questions de fond en traitement d'images : suivi de points identifiés, remplacement d'un contenu d'une vidéo, incrustation d'un modèle 3D... En bref, des problématiques riches à aborder dans le but de consolider nos bases en traitement d'images numériques.

En considérant le projet dans son ensemble, il a été important de considérer les diverses contraintes mais aussi les difficultés rencontrées. Pour le premier binôme, l'implémentation correcte du détecteur de Harris multi-échelles a présenté des difficultés de compréhension et de réalisation. Pour le second binôme, l'implémentation de l'homographie dans MATLAB a été l'élément le plus preneur. En général, la plus grande difficulté a été de détecter d'où venaient les erreurs. En effet, les temps de calcul de l'algorithme étant long, le débogage peut prendre beaucoup de temps. Dès l'implémentation des éléments graphiques, il a été tout de même tout de suite plus facile de comprendre le comportement de l'algorithme.

Annexes

---------- Fichier Main.m ----------

```
% Chargement des coordonnées des points suivi données par le fichier
pointsImage.m
load('cordonnees.mat', 'coord');

XEntier = coord(1:324,1:2:end);
YEntier = coord(1:324,2:2:end);

% Création de l'objet VideoWriter et ouverture
v = VideoWriter("videoTP");
open(v);

lune = imread("lune.jpeg");
vidObj = VideoReader('video_projet.mp4');

% Si les coordonnées de la main n'existent pas, les créer à partir d'une
frame de référence
frameIndex=100;
vidFrameForHand = read(vidObj, frameIndex);
image(vidFrameForHand);
vidObj.CurrentTime = 0;
firstX = XEntier(frameIndex,1:6);
firstY = YEntier(frameIndex,1:6);
if (~exist('XMain','var')||~exist('YMain','var'))
   [XMain, YMain] = SelectCadreMain(lune,firstX,firstY);
end

model = stlread("soucoupe5.stl"); % Lecture du modèle 3D
Seg = ConvertModelToSeg([0.5 0.5
0]+(model.Points)/30,model.ConnectivityList);

for i=1:324
   X = XEntier(i,1:6);
   Y = YEntier(i,1:6);
   vidFrame = read(vidObj, i);

vidFrame=Add3D(ChangeFeuilleBlanche(vidFrame,lune,X,Y,XMain,YMain),X,Y,S
eg);
writeVideo(v, vidFrame); % Ecriture de la frame sur la vidéo
end
% Fermeture de l'objet VideoWriter une fois la vidéo créée
close(v);
```

---------- Fichier GetH.m ----------

```
function H = GetH(x,y,X,Y)
nombreDePoint=min(length(x),length(X));
A=[];
B=[];
for i=1:nombreDePoint
   vector=[x(i) y(i) 1 0 0 0 -x(i)*X(i) -y(i)*X(i);
0 0 0 x(i) y(i) 1 -x(i)*Y(i) -y(i)*Y(i)];
   A=cat(1,A,vector);
   v2=[X(i);Y(i)];
   B=cat(1,B,v2);
end
h=A\B;
H=[h(1) h(2) h(3);
  h(4) h(5) h(6);
  h(7) h(8)  1 ];
end
```

---------- Fichier GetP.m ----------

```
function P = GetP(x,y,z,X,Y)
nombreDePoints = length(x);
A=[];
B=[];
for i=1:nombreDePoints
   vector=[
       x(i) y(i) z(i) 1 0 0 0 0 -x(i)*X(i) -y(i)*X(i) -z(i)*X(i);
       0 0 0 0 x(i) y(i) z(i) 1 -x(i)*Y(i) -y(i)*Y(i) -z(i)*X(i)];
   A=cat(1,A,vector);
   v2=[X(i);Y(i)];
   B=cat(1,B,v2);
end
p=A\B;
P=[p(1) p(2) p(3) p(4);
  p(5) p(6) p(7) p(8);
  p(9) p(10) p(11) 1];
end
```

```
            ---------- Fichier ConvertModelToSeg.m ----------
function Seg=ConvertModelToSeg(points,connectivityList)
   [n,~]=size(connectivityList);
   Seg=[];
   for i=1:n
       firstPointid=connectivityList(i,1);
       secondPointid=connectivityList(i,2);
       thirdPointid=connectivityList(i,3);
       Seg=[Seg;...
           points(firstPointid,:);points(secondPointid,:);...
           points(secondPointid,:);points(thirdPointid,:);...
           points(thirdPointid,:);points(firstPointid,:)];
   end
end


                  ---------- Fichier Add3D.m ----------
function newImage=Add3D(ancienneImage,X,Y,Seg)
alpha=2;
whiteSquareOffset=(double(2)/8)*0.2;
x=[0 1 1 0 0.125 0.5+whiteSquareOffset]*sqrt(2);
y=[0 0 1 1 0.6 0.5-whiteSquareOffset];
z=[0 0 0 0 0.2 0.3];
P=GetP(x,y,z,X,Y);
newImage=ancienneImage;
for i=1:2:length(Seg)-1
   premierVecteur = [Seg(i,1);Seg(i,2);Seg(i,3);1];
   deuxiemeVecteur= [Seg(i+1,1);Seg(i+1,2);Seg(i+1,3);1];
   vecteurDepart = P*premierVecteur;
   vecteurDepartx = round(vecteurDepart(1)/vecteurDepart(3));
   vecteurDeparty = round(vecteurDepart(2)/vecteurDepart(3));
   vecteurArrive = P*deuxiemeVecteur;
   vecteurArrivex = round(vecteurArrive(1)/vecteurArrive(3));
   vecteurArrivey = round(vecteurArrive(2)/vecteurArrive(3));
   X1=[vecteurDepartx vecteurDeparty];
   X2=[vecteurArrivex vecteurArrivey];

   newImage=DrawLineFat(X1,X2,newImage,alpha);
end
```

---------- Fichier ChangeFeuilleBlanche.m ----------

```
function
newImage=ChangeFeuilleBlanche(ancienneImage,imageFeuille,X,Y,XMain,YMain)

[h1,w1,~]=size(imageFeuille);
[h2, w2, ~] = size(ancienneImage);

x=[0;w1;w1;0];
y=[0;0;h1;h1];

H=GetH(X,Y,x,y); % Récupère la matrice d'homographie

newImage=ancienneImage;

for y2 = 1:h2
   for x2 = 1:w2

      M2 = [x2; y2; 1];
      M1 = H*M2;

      x1 = round(M1(1)/M1(3));
      y1 = round(M1(2)/M1(3));

      estDansImage = x1 >= 1 && x1 <= w1 && y1 >= 1 && y1 <= h1;

      % Booléen : True si le pixel est dans la zone encadrée pour la main
      estDansRectangle = x1 >= XMain(2) && y1 >= YMain(1) && y1 <= YMain(3);

estBleu=EstBleu([ancienneImage(y2,x2,1),ancienneImage(y2,x2,2),ancienneImage
(y2,x2,3)]);

   canWeEditPoint = estDansImage && (estBleu||~estDansRectangle);

   if (canWeEditPoint)
      % Place l'image
       newImage(y2,x2,1) = imageFeuille(y1,x1,1);
       newImage(y2,x2,2) = imageFeuille(y1,x1,2);
       newImage(y2,x2,3) = imageFeuille(y1,x1,3);
   end
   end
end
end
```

---------- Fichier DrawSeg.m ----------

```
% Récupérer la liste des points d'un segment fin
function I = DrawSeg(x1,x2,I,alpha)
   [h,w,~]=size(I);
   % Nombre de points à tracer par segment
   L = floor(alpha*sqrt((x2(1)-x1(1))^2+(x2(2)-x1(1))^2));
   % CoordX à tracer par segment
   %seg1 = zeros(L,2);
   for i=1:L
       %seg1(i,1)=round(x1(1)+i*(x2(1)-x1(1))/(L-1));
       %seg1(i,2)=round(x1(2)+i*(x2(2)-x1(2))/(L-1));
       x=round(x1(1)+i*(x2(1)-x1(1))/(L-1));
       y=round(x1(2)+i*(x2(2)-x1(2))/(L-1));
       estDansImage=x>1&&x<w&&y>1&&y<h;
       %[x,y]
       if (estDansImage)
           I(y,x,1)=244;
           I(y,x,2)=244;
           I(y,x,3)=221;
       end
   end
end
```

---------- Fichier DrawLineFat.m ----------

```
% Tracé d'un segment épais -> plusieurs segments fins
function I = DrawLineFat(X1,X2,I,alpha)
   dx = X2(1) - X1(1);
   dy = X2(2) - X1(2);
   val = sqrt(dx*dx+dy*dy);
   nX = -dy/val;
   nY = dx/val;
   for j=-1:1
       x1 = [(X1(1) + j*nX),(X1(2) + j*nY)];
       x2 = [(X2(1) + j*nX),(X2(2) + j*nY)];
       I = DrawSeg(x1, x2,I,alpha);
   end
end
```

---------- Fichier EstBleu.m ----------

```
function res=EstBleu(Color)
   couleurBleue=[116,158,171]; % Couleur de la feuille en RGB
   couleurBleueHSV=rgb2hue(couleurBleue); % On cherche la teinte de bleu
associée
   deltaAcceptable=0.35;
   ColorHSV=rgb2hue(Color); % On cherche la teinte associée à la couleur
de la main
   res= ColorHSV >= couleurBleueHSV * (1-deltaAcceptable)  ...
                    && ColorHSV<=couleurBleueHSV * (1+deltaAcceptable);
end
```

---------- Fichier pointsImage.m ----------

```
clear
close all
% Définition des constantes
n = 6;
sigmaG = 2;
sigmaC1 = 3;
sigmaC2 = 5;
fenetreW = 37;

% Lecture de la vidéo
vidObj = VideoReader("video_projet.mp4");
vidObj.CurrentTime =0;
currAxes = axes;

% Lecture 1ère frame pour récupérer les n points
vidFrame = readFrame(vidObj);
imshow(vidFrame);

% Récupération des n coins
coordonnesGInput = ginput(n);
coordInit = Initialisation(coordonnesGInput,n);

% Récupération des coordonnées des coins
coord = RecupCoins(vidObj,sigmaG,sigmaC1,sigmaC2,n,coordInit);

% Points que l'on envoie au groupe II
x = coord(1:vidObj.NumFrames,1:2:end);
y = coord(1:vidObj.NumFrames,2:2:end);
% Fin algo
%-----------------------------------------------------------------------%
```

```
% Fonctions
function D = Harris(sigmaG,sigmaC1,sigmaC2,image)

   % Initialisation

   [X1,Y1] =
meshgrid(ceil(-sigmaC1*3:sigmaC1*3),ceil(-sigmaC1*3:sigmaC1*3));
   [X2,Y2] =
meshgrid(ceil(-sigmaC2*3:sigmaC2*3),ceil(-sigmaC2*3:sigmaC2*3));
   [XG,YG] = meshgrid(ceil(-sigmaG*3:sigmaG*3),ceil(-sigmaG*3:sigmaG*3));


   % POUR D1
   % Calcul de la Gausienne verticale et horizontale
   Gx=-XG.*exp(-(XG.^2+YG.^2)/(2*sigmaG^2))/(2*pi*sigmaG^4);
   Gy=-YG.*exp(-(XG.^2+YG.^2)/(2*sigmaG^2))/(2*pi*sigmaG^4);
   G1= exp(-(X1.^2+Y1.^2)/(2*pi*sigmaC1^2))/(2*pi*sigmaC1^2);

   % Normalisation gausiennes
   Gx= Gx/(sum(sum(-XG.*Gx)));
   Gy= Gy/(sum(sum(-YG.*Gy)));
   G1 = G1/(sum(sum(G1)));

   % Calcul du gradient Ix, et Iy en tout point
   Ix=conv2(image,Gx,'same');
   Iy=conv2(image,Gy,'same');

   % Calcul de l'ensemble des covariances
   Cxx1 =conv2((Ix.*Ix),G1,'same');
   Cyy1=conv2((Iy.*Iy),G1,'same');
   Cxy1=conv2((Ix.*Iy),G1,'same');

   % POUR D2
   % Calcul de la Gausienne verticale et horizontale
   G2= exp(-(X2.^2+Y2.^2)/(2*pi*sigmaC2^2))/(2*pi*sigmaC2^2);

   % Normalisation gausiennes
   G2 = G2/(sum(sum(G2)));




% Calcul de l'ensemble des covariances
```

```
    Cxx2=conv2((Ix.*Ix),G2,'same');
    Cyy2=conv2((Iy.*Iy),G2,'same');
    Cxy2=conv2((Ix.*Iy),G2,'same');

    % Calcul de détecteur de Harris
    D1 = Cxx1.*Cyy1 -Cxy1.*Cxy1 - 0.05.*(Cxx1+Cyy1).^2;
    D2 = Cxx2.*Cyy2 -Cxy2.*Cxy2 - 0.05.*(Cxx2+Cyy2).^2;
    D = min(D1*norm(D2),norm(D1)*D2);
end
%-------------------------------------------------------------------------%
function coordMax = coordMaxCalcul(coordonnes,thImage)
    maxi = -1e-30;
    for c = coordonnes(1):coordonnes(1)+37
        for d = coordonnes(2):coordonnes(2)+37

            if thImage(d,c)>maxi
                maxi = max(maxi,thImage(d,c));
                coordMax = [c d];
            end
        end
    end
end
%-------------------------------------------------------------------------%
function maxList = CreateList (coordonnes,thImage,n)
        maxList = zeros(1,2*n);
        % Initialise une liste de maximum vide

    for l = 1:n % Parcourt les 6 points

        % Appelle la fonction coordmaxcalcul sur les 4 points de la liste
        coords = coordMaxCalcul(coordonnes((2*l-1):(2*l)),thImage);

        maxList(2*l-1) = coords(1);
        maxList(2*l) = coords(2);
    end
end
%-------------------------------------------------------------------------%
function coordInit = Initialisation(coordonnesGInput,n)
    coordInit = zeros(1,2*n);
    for i = 1:n
    coordInit(2*i-1) = coordonnesGInput(i,1);
    coordInit(2*i) = coordonnesGInput(i,2);
    end
end
function coord = RecupCoins(vidObj,sigmaG,sigmaC1,sigmaC2,n,coordInit)
```

```
    coord = zeros(vidObj.NumFrames,2*n);
    for i = 1:vidObj.NumFrames

        % Extraction de la frame
        vidFrame = read(vidObj,i);

        % Passage en luminance  de l'image extraite en rgb/pixel
        newImage = 0.299*vidFrame(:,:,1) +
0.587*vidFrame(:,:,2)+0.114*vidFrame(:,:,3);

        % Application du détecteur de Harris multiEchelle
        thImage = Harris(sigmaG,sigmaC1,sigmaC2,newImage);
        if i == 1

        % Fenetrage
            coordInit = coordInit - 19 ;
            coordInit = uint16(coordInit);
            coinsCord = CreateList(coordInit,thImage,n);
            coord(i, :) = coinsCord;

        elseif i == 2

            % Deuxième itération
            coordSecondInit = coord(i-1, :)-19;
            coordSecondInit = CreateList(coordSecondInit,thImage,n);
            coord(i, :) = coordSecondInit;
        else

            % Toutes les autres itérations
            coordInitNext = zeros(8);
            for p = 1:n
                coordInitNext(2*p-1) =
uint16(coord(i-1,2*p-1)+(coord(i-1,2*p-1)-coord(i-2,2*p-1))/2);
                coordInitNext(2*p) =
uint16(coord(i-1,2*p)+(coord(i-1,2*p)-coord(i-2,2*p))/2);
            end
            coordInitNext = coordInitNext-19;
            coordInitNext = CreateList(coordInitNext,thImage,n);
            coord(i, :) = coordInitNext;
        end
    end
end
```

---------- Fichier rgb2hue.m ----------

```
function [H] = rgb2hue(RGB)
   isColorAnInteger=RGB(1)>2;
   if(isColorAnInteger)
       RGB=double(RGB)/255;
   end
   Cmax = max(RGB);
   Cmin = min(RGB);
   delta = Cmax - Cmin;

   R = RGB(1);
   G = RGB(2);
   B = RGB(3);

   if delta==0
       H = 0;
   elseif Cmax == R
       H = 60*(mod((G-B)/delta,6));
   elseif Cmax == G
       H = 60*(((B-R)/delta) + 2);
   elseif Cmax == B
       H = 60*(((R-G)/delta) + 4);
   end
end
```

---------- Fichier SelectCadreMain.m ----------

```
function [XMain,YMain]=SelectCadreMain(imageFeuille,X,Y)
[h1,w1,~]=size(imageFeuille);
x=[0;w1;w1;0];
y=[0;0;h1;h1];
H=GetH(x,y,X,Y);
[XMain, YMain] = ginput(4);
for i=1:4
    M2 = [XMain(i); YMain(i); 1];
    M1 = H\M2;
    XMain(i) = round(M1(1)/M1(3));
    YMain(i) = round(M1(2)/M1(3));
end
end
```