Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

2022/2023

ITIV

Project report

Mosaicing-plan (5)

Ecole	Haute école d'ingénierie et d'architecture de Fribourg (HEIA-FR)
Cours	Introduction au traitement d'image et à la vision 3D
Elèves	Barras Simon, Rojas David, Steiger Damien
Classe	ISC-IL-3
Professeur	Chabbi Houda
Date	30.01.2023

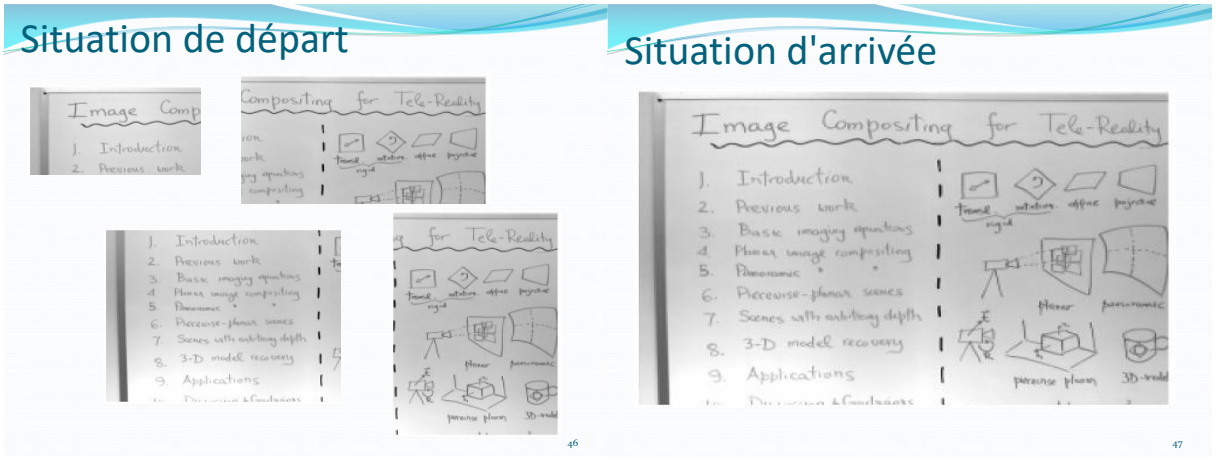
Hes·so FRIBOURG/FREIBURG
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz

1 Introduction

Dans le cours d'ITIV nous avons vu plusieurs outils qui permettent de manipuler des images ou mesurer des grandeurs physiques. Afin de s'exercer, nous avons dû réaliser un mini-projet comportant l'un des deux outils. La première implémentation utilisant les méthodes d'OpenCV et une seconde en calculant à la main à l'aide de la théorie vu en cours.

Nous avons choisi le thème numéro 5 qui consiste à reconstruire un plan à partir d'au minimum 3 images. Cette reconstruction s'appelle le « mosaicing-plan » et nous aurons besoin de l'homographie.

Voici un exemple de mosaicing-plan :



1.1 Objectifs

Le but de notre projet est d'immortaliser le tableau de la salle D20.18 personnalisé par ses élèves. Pour ne pas perdre la qualité des dessins qui s'y trouve, nous voulons numériser ce tableau au grand format à l'aide de 3 petites images.

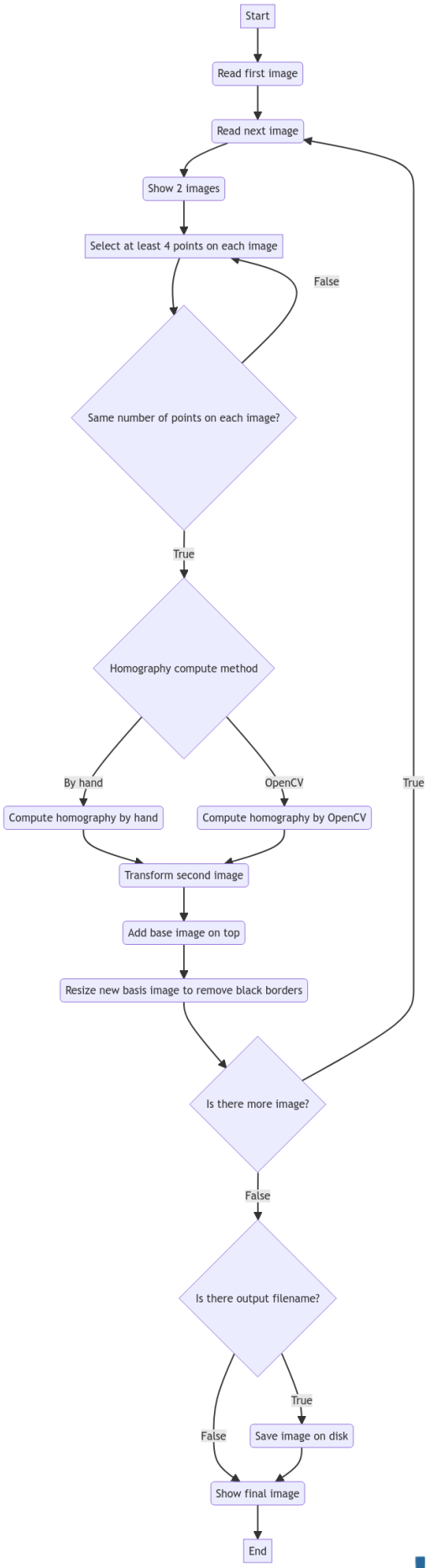
Les objectifs sont les suivants :

- Lire n images d'un plan (au minimum 3)
- Créer des pairs de points pour associer les 2 images
- Calculer l'homographie avec OpenCV et à la main
- Construire l'image mosaïque

2 Fonctionnalités

Notre programme a une unique fonctionnalité qui permet de fusionner des images afin de recréer un plan. Cet outil s'utilise avec la ligne de commande et est configurable à l'aide des paramètres. Il est notamment possible de sélectionner les photos voulues, la méthode de calcul de l'homographie et le fichier de sortie.

Le schéma suivant explique le fonctionnement de l'application :


Start
Read first image
Read next image
Show 2 images
Select at least 4 points on each image
False
Same number of points on each image?
True
Homography compute method
By hand
OpenCV
Compute homography by hand
Compute homography by OpenCV
Transform second image
Add base image on top
Resize new basis image to remove black borders
Is there more image?
True
False
Is there output filename?
True
False
Save image on disk
Show final image
End

3 Implémentation

Nous avons réalisé notre code en python en s'aidant des outils de la librairie « OpenCV ».

3.1 OpenCV

Nous avons grandement utilisé OpenCV pour construire notre application. Voici la liste des méthodes que nous avons utilisé avec une explication de leur but et de leurs paramètres. Tous les bout de codes OpenCV de notre projet sont commentés.

```python
# import opencv
import cv2

# draw a line ((x;y) point de départ, (x;y) point d'arrivée, couleur, épaisseur)
cv2.line(image, (x1, y1), (x2, y2), color, THICKNESS)

# Show image (window_name, image)
cv2.imshow(name, img)

# Transformation de perspective (image, homographie, dimensions de l'image)
image_out = cv2.warpPerspective(image, homography, (width, height))

# Convertir l'image en niveaux de gris (image, convertisseur)
image_gray = cv2.cvtColor(imgage, cv2.COLOR_BGR2GRAY)

# Trouver les pixels non noirs (image)
coords = cv2.findNonZero(image)

# Obtenir les coordonnées du rectangle (coordonnées)
x, y, width, height = cv2.boundingRect(coords)

# Trouve la matrice de transformation (Liste des points à transformer, liste des points de destination)
homography, status = cv2.findHomography(point_list1, point_list2)

# Wait for a key to be pressed (time in ms)
c = cv2.waitKey(1)

# Add black pixels around the image
# (image, top margin, bottom margin, left margin, right margin, type of border, color of the border)
image_border =  cv2.copyMakeBorder(image, top_margin, bottom_margin, left_margin, right_margin,
cv2.BORDER_CONSTANT, value=[0, 0, 0, 0])

# Create a window with a name (window name, window type)
cv2.namedWindow(name, cv2.WINDOW_NORMAL)

# Set the mouse callback (window name, callback function, callback parameter)
cv2.setMouseCallback(name, my_mouse_callback, param)

# Display the image (window name, image)
cv2.imshow(name, imgage)

# Read the images (image path, image type)
image = cv2.imread(image_file, cv2.IMREAD_COLOR)

# Close all the windows
cv2.destroyAllWindows()

# Write the image (image path, image)
cv2.imwrite(output_file, image
```

3.2 Homographie à la main

Pour trouver la matrice d'homographie, nous avons utilisé la formule suivante :

$$m = (A^T * A)^{-1} * A * B$$

La matrice *m* étant la version 1D de la matrice h. La matrice *B* est les points cibles, dans notre cas, les points de *cp_base*. La matrix *A* représente un système d'équation, elle se construit de la manière suivante :

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & -x_i u_i & 1 & -y_i u_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -x_i v_i & -y_i v_i \end{bmatrix}$$

Cette matrix a donc 2**nombre de points* lignes et 8 colonnes.

Le code suivant construit les matrix nécessaires puis résout l'équation à l'aide de « numpy » :

```python
def find_my_own_homography(cp_add, cp_base):
    """
    Compute the homography with a couple of points
    :param cp_add: points of the image to add
    :param cp_base: points of the image base
    :return: the homography matrix
    """
    # transform 2D array to 1D array
    target = cp_base.reshape(-1, 1).ravel()
    # format cp_add to the Matrix A
    A = np.zeros((len(cp_add) * 2, 8))
    for i in range(len(cp_add)):
        A[2 * i, 0:2] = cp_add[i]
        A[2 * i, 2] = 1
        A[2 * i, 6:8] = -cp_add[i] * target[2 * i]
        A[2 * i + 1, 3:5] = cp_add[i]
        A[2 * i + 1, 5] = 1
        A[2 * i + 1, 6:8] = -cp_add[i] * target[2 * i + 1]
    # compute the homography m = (AT*A)^-1 * A * target
    m = np.dot(np.dot(np.linalg.inv(np.dot(A.T, A)), A.T), target)
    # reshape m to the homography matrix
    H = np.array([[m[0], m[1], m[2]], [m[3], m[4], m[5]], [m[6], m[7], 1]])
    return H
```

3.3 Arguments

Dans notre programme, nous avons ajouté la gestion des paramètres afin de configurer l'expérience.

Nous nous sommes aidé la librairie « argparse ». Cette librairie permet de parser automatiquement les paramètres ainsi qu'afficher. Voici l'utilisation de notre librairie dans la main :

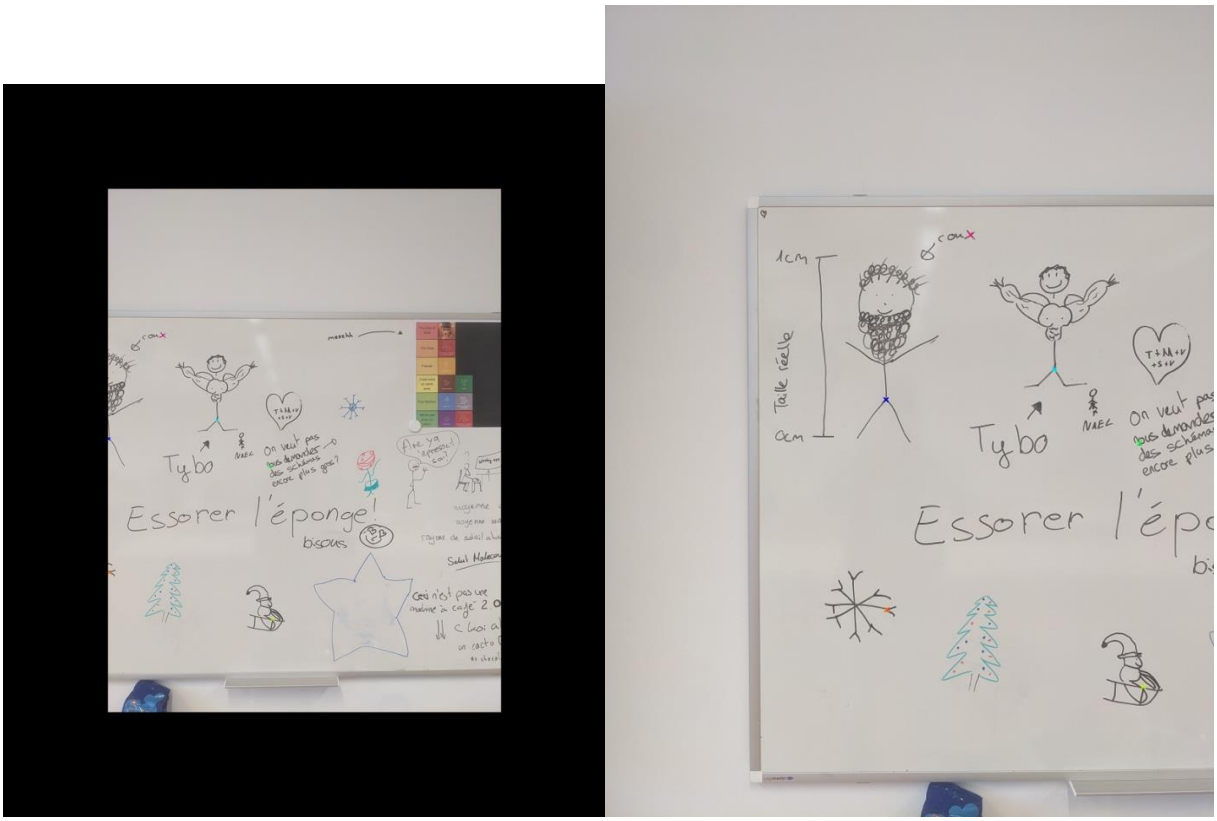
```python
# Define arguments
parser = argparse.ArgumentParser(description='Tool to merge images together')
parser.add_argument('-c', '--custom', default=False, action='store_true', help='Use our homography computation instead of opencv')
parser.add_argument('-d', '--directory', default="./", type=str, help='Directory containing the images to merge')
parser.add_argument('-o', '--output', default=None, type=str, help='Output file name')
# Images as parameters
parser.add_argument('images', type=str, nargs='*', help='Images to merge')

# Parse arguments
args = parser.parse_args()

# If no images are given, print help
parser.print_help()

# Build the list of images
imgs = []
if args.images:
    for i in args.images:
        imgs.append(args.directory + "/" + i)
else:
    imgs = sorted(glob.glob(args.directory + "/*.jpg"))

# Call the main function
main(imgs, args.custom, args.output)
```

Voici le résultat lorsque nous demandons l'aide :

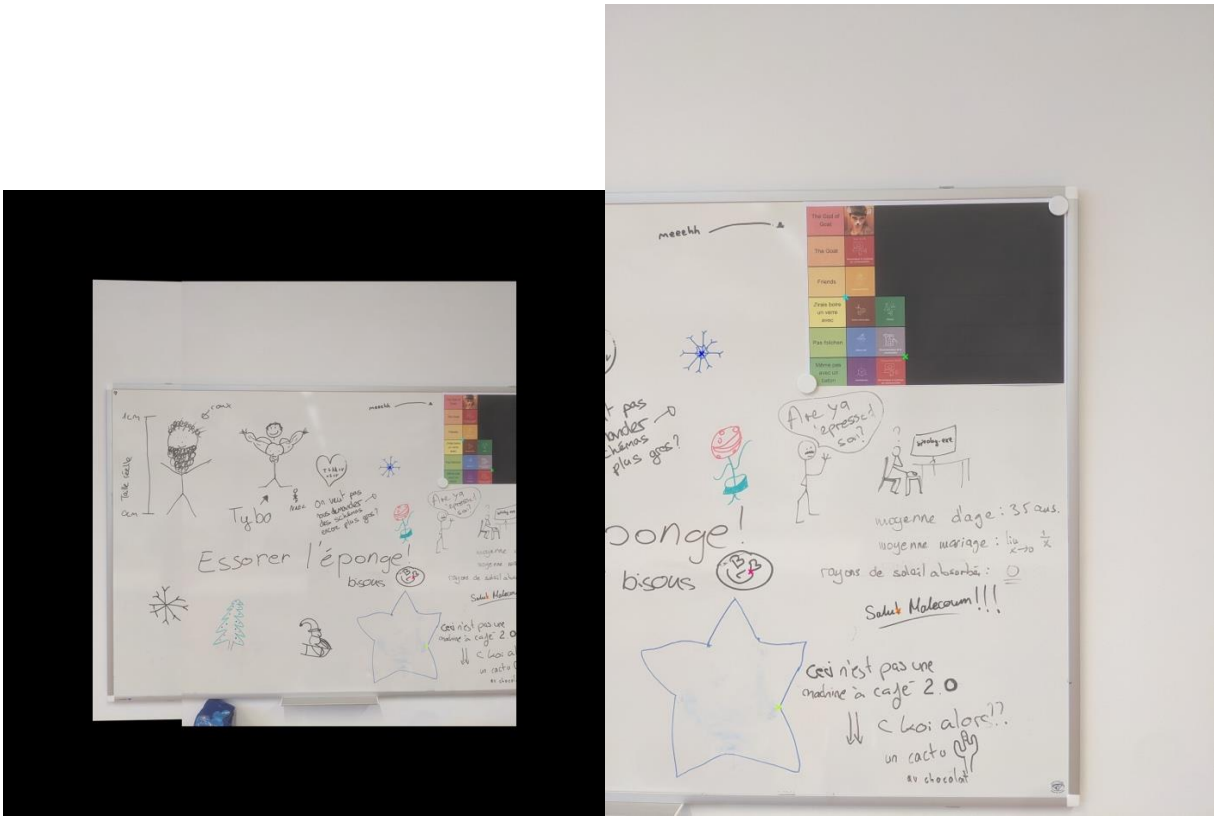
```
simon on Exoscale() at …/ITIV_2023_Gr_7 on develop ( ) via (venv)
  python .\tp2\main.py -h
usage: main.py [-h] [-c] [-d DIRECTORY] [-o OUTPUT] [images ...]

Tool to merge images together

positional arguments:
  images                Images to merge

options:
  -h, --help            show this help message and exit
  -c, --custom          Use our homography computation instead of opencv
  -d DIRECTORY, --directory DIRECTORY
                        Directory containing the images to merge
  -o OUTPUT, --output OUTPUT
                        Output file name
```

4 Résultats obtenus

Sélection des points : Image centre – Image gauche



Sélection des points : Image centre – Image droite

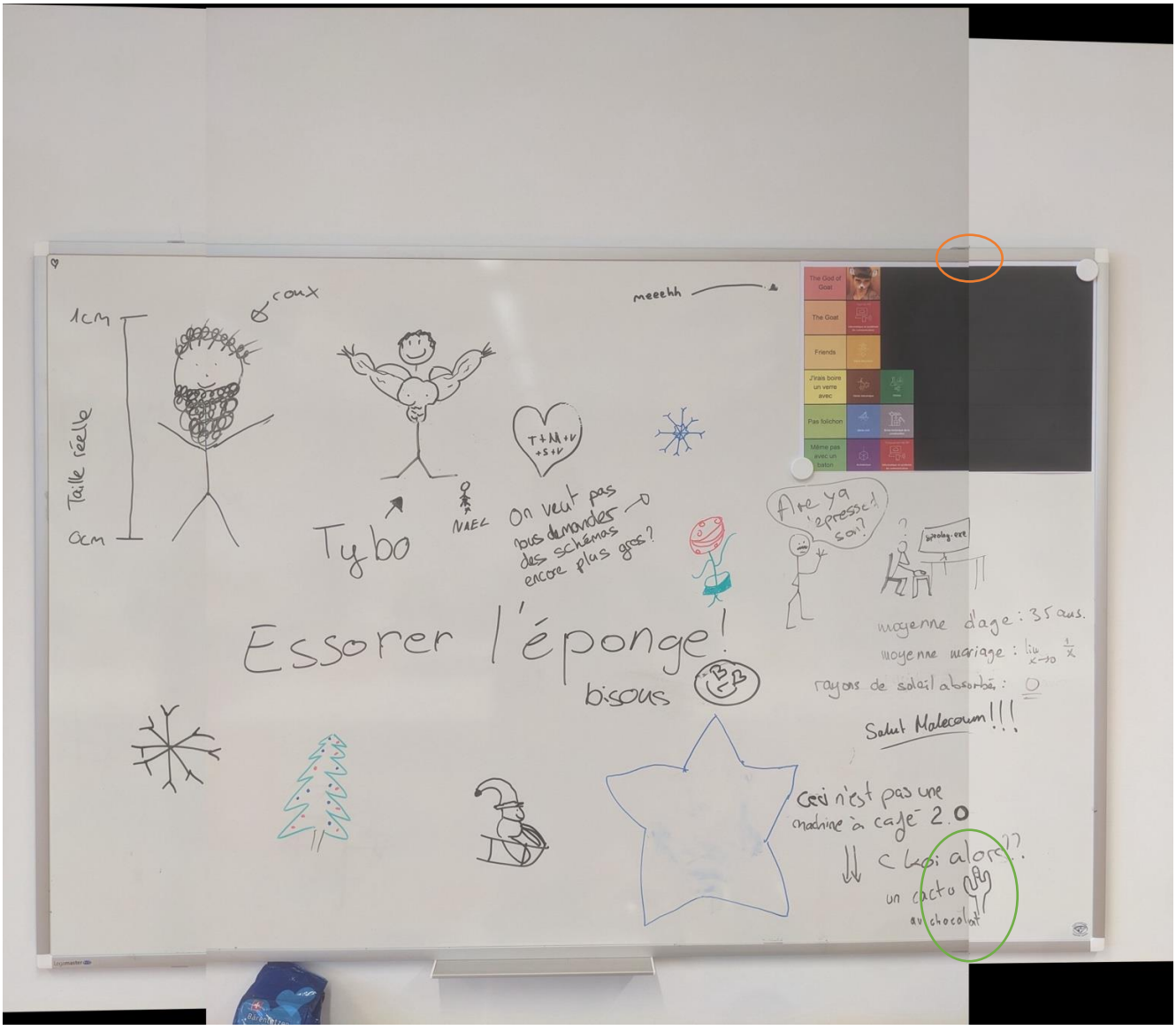


4.1 Comparaison

La comparaison a été réalisé avec l'aide de la méthode/fichier démo, donc les mêmes exacts points.

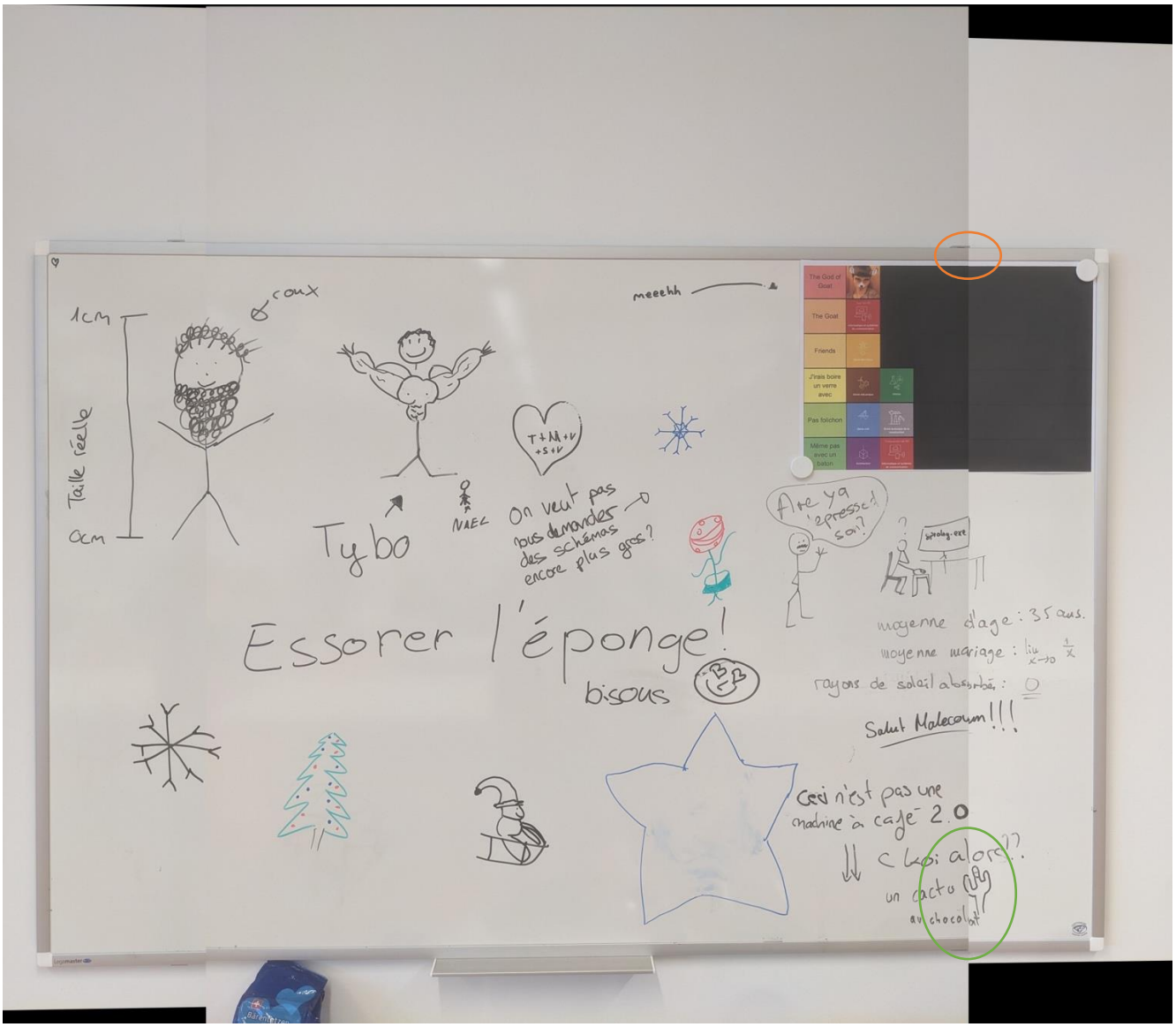
cv2.findHomography()

- En orange : On aperçoit un léger décalage entre l'impression sur l'image du centre et celle de droite (→ un peu surélevé).
- En vert : La fin du 'l' de 'alors' est parfaitement écrit et le cactus suit sa forme initiale.



Notre méthode

- En orange : On n'aperçoit quasiment pas de décalage entre l'impression sur l'image du centre et celle de droite (→ pas surélevé).
- En vert : On distingue très rapidement que le cactus est un peu décalé.



Conclusion

On peut apercevoir, soit un léger décalage sur le haut du Mosaic-plan avec la méthode d'OpenCV, soit un décalage plus prononcé sur le bas du Mosaic-plan avec notre propre méthode.

On peut donc en déduire que les points choisis pour fusionner la dernière image (celle de droite) n'étaient pas parfait, mais que l'on a tout de même une divergence dans la manière de calculé l'homographie, vraisemblablement impacté par le nombre de point sélectionné.

5 Mode d'emploie

Pour lancer l'application, il faut utiliser certains paramètres :

Arguments	Obligatoire	Description
-h	Non	Afficher l'aide
-c / --custom	Non	Si présent la méthode de calcul sera celle que nous avons fait nous même à la main
-d / --directory	Non	Chemin jusqu'au dossier qui contient les images à fusionner. Permet d'éviter de le remettre à chaque image.
-o / --output	Non	Représente le fichier dans lequel l'image finale sera enregistrée. S'il n'est pas présent, alors l'image ne sera pas enregistrée.
Images	Oui	Liste des images à fusionner. La première image est l'image de base à laquelle les autres seront ajoutées. Il est important de bine la choisir pour éviter de répercuter du bruit. L'ordre des images données est celui qui sera utilisé par l'application pour les fusionner

Une fois que l'application est lancée, il suffit de sélectionner les points sur les images présentées. L'index des points est important en revanche, l'ordre de l'image sélectionnée n'est pas important. Une fois que les points ont été sélectionnés, il faut appuyer sur « **v** » afin de valider la sélection ou « **r** » pour recommencer la sélection. La touche « **ESC** » permet de quitter la fusion.

6 Sources

Une partie du code a pu être récupéré du TP1, quelques petites modifications ont été faites ici et là, afin que le code soit plus en accord avec le résultat voulu par le groupe.

Nous avons utilisé la documentation de la librairie OpenCV (https://docs.opencv.org/4.x/) afin de voir les méthodes disponibles et le détail des paramètres.

Nous nous sommes aussi servi du forum officiel lorsque nous avions eu des problèmes similaires à d'autre personne (https://answers.opencv.org/questions/)

Mention spéciale à l'outil « Github copilot » qui nous a aidé à coder cette application

7 Conclusion

Pour conclure, nous avons bien aimé ce projet car le travail ne ressemblait à aucun autre. En effet, le traitement d'image n'est pas quelque chose que nous avions déjà vu et c'est un thème spécifique de l'informatique que nous n'abordons pas du tout lors de notre cursus. Nous avons aussi trouvé que la taille du projet était bien adaptée et OpenCV permet de rapidement obtenir un résultat.

Finalement, nous avons bien réussi à se partager le travail.