Documentation Technique
Programme d'assemblage d'images

Prérequis

Le programme est réalisé avec le langage python. Pour que le programme fonctionne il faut utiliser la version 3.11 (ou ultérieure) de python. Il faut aussi avoir les extensions python suivantes :
- numpy
- opencv
- skimage ou scikit-image

Les images doivent tout être au format .png pour le programme d'assemblage comme celui des métriques.
Le code source du programme est accessible depuis ce dépot git :
https://gitlab.com/a.allies/projet-math-info

Structure du programme

Le dossier qui contient le programme est structuré de la manière suivante :
Le dossier *src* contient tout les fichiers python nécessaires au programme d'assemblage des images et celui des métriques.
Pour le bloc Le dossier *bloc* contient les résultats des différents blocs de l'algorithme :
Les images avec leurs points clés sont enregistrées dans le fichier *image_gauche_p_cles.png* et *image_droite_p_cles.png*.
On a les 2 images avec leurs correspondances enregistrées dans le fichier *img_matches.jpg* ;
La matrice homographique enregistrée dans le fichier *matrice_homographique.txt*.
On a l'image de droite à qui on a appliqué la transformation enregistrée dans le fichier *warped_img.jpg*
Le dossier *images* est le dossier qui doit contenir les images pour le programme d'assemblage.
Le dossier *image_metriques* est celui qui doit conternir l'image pour la programme des metriques.

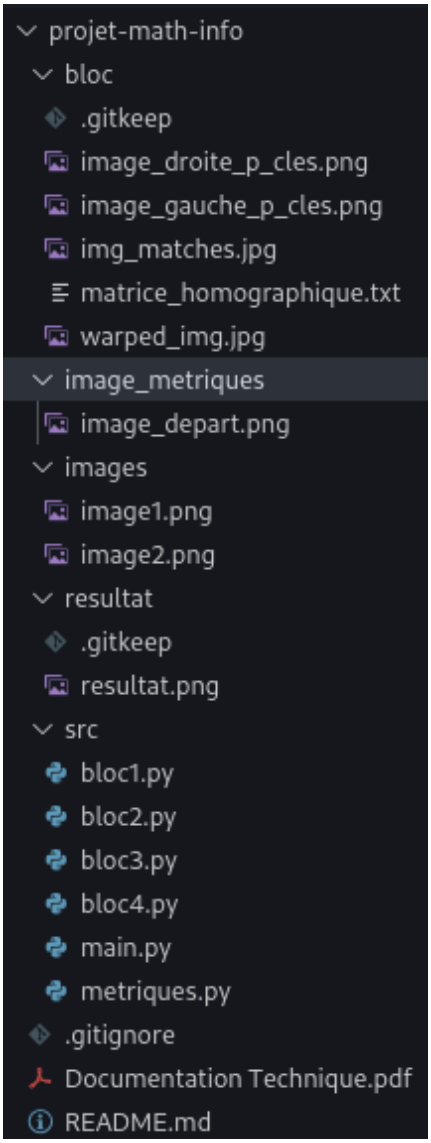

Figure 1 – Structure du programme

Utilisation du programme

Pour utiliser le programme d'assemblage d'images, il faut placer les images à assembler dans le dossier images du dossier src. Il faut nommer l'image la plus 'a gauche image1.png, celle immédiatement à sa droite image2.png, etc ...
Ensuite il faut lancer le fichier main.py L'image résultat de l'algorithme se trouve dans le dossier résultat au .png.
Pour les calculer les métriques SSIM et PSNR il faut placer une image, nommée image depart.png, dans le dossier *images_metriques*. Il faut ensuite lancer le fichier metriques.py.

main

- View Source

```
def programme_assemblage(gaussian_blur=True) -> None:
```

- View Source

C'est la fonction principale du programme d'assemblage d'images. D'abord la fonction va chercher toutes les images dans le répertoire images. Ensuite elle les redimensionne si nécessaire, et enregistre leur adresse dans le tableau des adresses des images. Ensuite la fonction boucle sur les images du tableau des images : Elle fusionne les 2 premières, puis la 3ème avec la fusion de la 1ère et la 2ème, etc ... La fonction enregistre l'image résultat dans le dossier resultat.

bloc1

• View Source

def redimension_et_blur(adresse_image: str, nb_image: int, gaussian_blur=**True**) -> **None**: • View Source

Cette fonction reçoit en argument l'adresse d'une image, un entier et un booléen. Si cette image contient 1 milion de pixels ou plus elle est redimensionnée, réduite, tout en prenant en compte le rapport de proportionalité entre la longueur et la largeur de cette image : Si elle plus large que longue, etc ... Le booléen représente si on doit appliquer ou non le flou gaussien. La fonction enregistre enfin l'image dans le répertoire images de l'algorithme d'assemblage d'images. L'argument de type entier de la fonction permet d'enregistrer l'image avec le bon numéro (pour l'ordre d'assemblage).

def p_cles_et_descripteurs(image: numpy.ndarray) -> tuple[tuple[cv2.KeyPoint], numpy.ndarray]: • View Source

Cette fonction prend en argument une image. La fonction uilise la fonctionSIFT d'open cv2 pour renvoyer les points clés et descripteurs de cette dernière.

bloc2

• View Source

```
def FindMatches(
        BaseImage_descrpt: numpy.ndarray,
        SecImage_descrpt: numpy.ndarray
) -> list[list[cv2.DMatch]]:
```

• View Source

La fonction prend en argument les descripteurs de 2 images. La fonction crée une objet de type <> pour trouver toutes les correspondances entre ces 2 images. On boucle ensuite sur ces correspondances pour trouver les meilleures : Si la distance de la meilleure correspondance m est inférieure à 0.75 fois la distance de la deuxième meilleure correspondance n c'est une bonne correspondance. Enfin la fonction retourner une liste contenant les meilleures correspondances.

bloc3

• View Source

```
def FindHomography(
    Matches: list[list[cv2.DMatch]],
    BaseImage_kp: tuple[cv2.KeyPoint],
    SecImage_kp: tuple[cv2.KeyPoint]
) -> tuple[numpy.ndarray, numpy.ndarray]:
```

• View Source

Fonction pour trouver la matrice d'homographie entre deux ensembles de points correspondants.

Arguments :

- Matches : Liste des correspondances de points entre les deux images
- BaseImage_kp : Points clés de l'image de référence
- SecImage_kp : Points clés de l'image secondaire

Retourne :

- HomographyMatrix : Matrice d'homographie calculée
- Status : Statut de la correspondance

```
def getNewFrameSizeAndMatrix(
    HomographyMatrix: numpy.ndarray,
    Sec_ImageShape: tuple[int, int, int],
    Base_ImageShape: tuple[int, int, int]
) -> tuple[list[int, int], list[int], tuple[numpy.ndarray[numpy.ndarray[numpy.float32]]]]:
```

• View Source

Fonction pour calculer la taille de la nouvelle image résultante après transformation homographique et mettre à jour la matrice d'homographie en conséquence.

Arguments :

- HomographyMatrix : Matrice d'homographie
- Sec_ImageShape : Forme de l'image secondaire
- Base_ImageShape : Forme de l'image de référence

Retourne :

- Nouvelle taille de l'image
- Correction à appliquer à la matrice d'homographie
- Matrice d'homographie mise à jour

bloc4

View Source

```
def images_avec_correspondances(
    matches: list[list[cv2.DMatch]],
    image1: numpy.ndarray,
    image2: numpy.ndarray,
    pcles1: tuple[cv2.KeyPoint],
    pcles2: tuple[cv2.KeyPoint]
) -> None:
```

View Source

Cette fonction prend en paramètre les correspondances, 'matches' et les points clés, pcles1 et pcles2, entre image1 et image2. D'abord la fonction met les deux images l'un à coté de l'autre , puis cherche les points commun et relie ces derniers. Cela permet de vérifier visuellement les points communs des deux images. La fonction retourne une image sur lequel on voie les deux images, avec leurs points communs et des droites qui relient ces deniers.

```
def assemblage(
    BaseImage: numpy.ndarray,
    SecImage: numpy.ndarray,
    BaseImage_kp: tuple[cv2.KeyPoint],
    SecImage_kp: tuple[cv2.KeyPoint],
    descriptors1: numpy.ndarray,
    descriptors2: numpy.ndarray
) -> numpy.ndarray:
```

View Source

La fonction assemblage prend en paramètre deux images, leur points clés et leur descripteurs. La fonction considère la première image comme étant l'image de base puis elle fait appel à la fonction Matches pour trouver les correspondances entre les 2 images. Ensuite la fonction calcule la matrice homographique. Puis la fonction getNewFrameSizeAndMatrix estime une nouvele taille pour l'image finale et une nouvelle matrice homographique. Enfin la fonction warpPerspective d'open cv2 applique la matrice homographique sur la deuxième image puis colle l'image de base à l'image obtenu avec la matrice homographique. La fonction renvoie une image composée des deux images précédentes bien positionnées.

metriques

• View Source

def image_rotation(image: numpy.ndarray, angle=-30) -> numpy.ndarray: • View Source

Cette fonction prend en argument et un angle, x. Avec l'angle et une fonction d'open cv2 on calcule une matrice de rotation. Enfin la fonction renvoie l'image de départ qui a subit une rotation de x degrés.

def deux_images(valeur_superposition: float) -> tuple[numpy.ndarray, numpy.ndarray]: • View Source

La fonction va chercher l'image dans le dossier image_metriques. La fonction prend un seul argument la valeur de superposition. On calcule l'axe vertical central de l'image, une colonne centrale. La première image est découpée de la colonne de début à la colonne centrale, l'axe vertical central, plus la superposition. La seconde image est découpée de la colonne centrale moins la superposition à la fin de l'image. La fonction retourne ces deux images.

def psnr(image1: numpy.ndarray, image2: numpy.ndarray) -> float: • View Source

Calcul du PSNR Cette fontion prend en paramètre deux images. Elle vérifie si les deux images ont la même taille. Si oui on calcule le psnr, sinon on redimensionne la deuxième image puis on calcule le psnr. Enfin elle renvoie le psnr.

def ssim(image1: numpy.ndarray, image2: numpy.ndarray) -> float: • View Source

Cette fonction prend deux images et renvoie le ssim entre ces deux images.

def programme_metrique(valeur_superposition: float, angle: int, gaussian_blur=True) -> None: • View Source

Cette fonction permet de mesurer le bruit que l'algorithme d'assemblage d'images ajoute à une image, image_depart. Elle a 3 arguments, un premier est la valeur de superposition que l'on choisit pour les deux images que l'on va obtenir à partir d'image départ. Le troisième argument permet de chosir si on veut ou pas appliquer une flou gaussien dans l'algorithme d'assemblage. Pour cela on calcule d'abord les deux images à partir de image_depart. Ensuite on fait 3 tests. Pour le premier on enregistre les deux images dans le dossier images de l'algorithme d'assemblage, on applique ce dernier. On calcule le ssim et le psnr entre image_depart et le resultat de l'algorithme d'assemblage. Pour le deuxième test on fait pareil mais la seconde image a subit une rotation de x degrés dans le sens horaire. x étant le deuxième argument de la fonction. Pour le troisème test on fait pareil mais la seconde image a subit une rotation de x degrés dans le sens antihoraire.