TP 5 - Création de panoramiques

Les fichiers suivants sont à rendre pour ce TP :

— un fichier **PDF** avec les copies d'écrans nécessaires, les commentaires demandés et les résultats obtenus,
— les fichiers sources.

1 INTRODUCTION

Le but de ce TP est de vous initier à plusieurs problématiques importantes de vision par ordinateur :

— La détection de points caractéristiques (ou points d'intérêt, aussi appelés *feature points*),
— la description de ces points caractéristiques,
— la mise en correspondance (aussi appelé appariement) de points d'intérêt détectés dans plusieurs images,
— le calcul d'une homographie,
— la réalisation d'un collage d'images (*stitching*, voir Figure 1.1) en vue de réaliser un panoramique.

Il est relativement simple d'implémenter un programme prenant en entrée 2 (ou plus, cela vous est laissé en exercice) images pour les "coller" à partir du moment où elles ont suffisamment d'informations communes.
L'idée de l'application est la suivante :

1. charger les deux images (une sera l'image "gauche" – `G` et l'autre l'image "droite" – `D`;
2. détecter dans chacune des images un ensemble de points d'intérêt FP_g et FP_d en utilisant un de nombreux détecteurs fournis par OpenCV;
3. calculer pour chacun des points de FP_g (respectivement FP_d) un vecteur descripteur (aussi appelé descripteur tout court) FD_g (resp. FD_d).

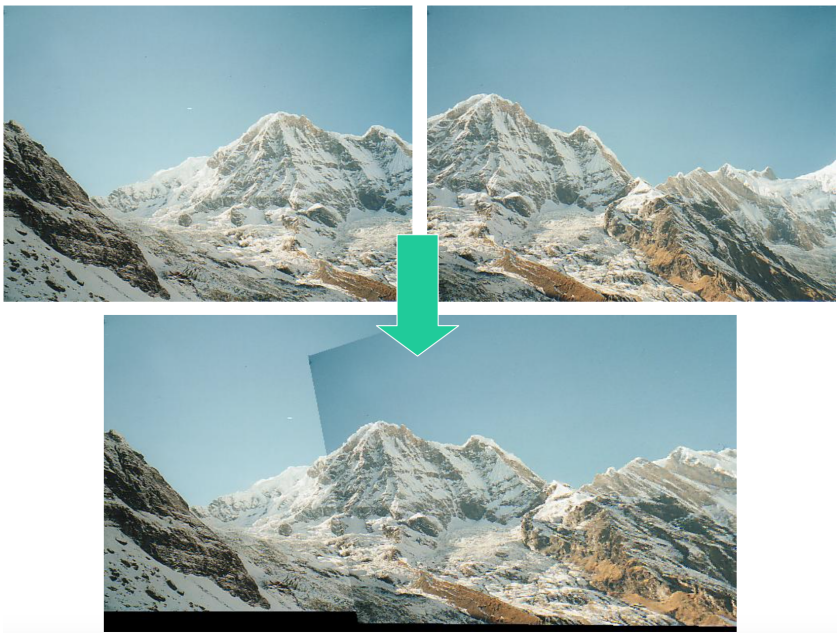

FIGURE 1.1 – Objectif final du TP.

4. Essayer de faire l'appariement (i.e. de mettre en correspondance) FD_g et FD_d afin de trouver "quels points de l'image gauche correspondent à ceux de l'image droite" ;
5. calculer la transformation géométrique permettant de passer des points de l'image `G` à l'image `D` : c'est une homographie car nos deux ensembles de points sont planaires ;
6. appliquer cette transformation géométrique à l'image `D` pour obtenir D_H ;
7. "coller" l'image D_H à côté de l'image `G`.

Nous allons brièvement expliquer les différentes étapes dans la suite et nous vous fournissons un squelette d'application Python/OpenCV ainsi que quelques images permettant de réaliser ce collage d'images.

1.1 DÉTECTION DE POINTS D'INTÉRÊT

L'idée de la détection de points d'intérêt (ou plus généralement de "zones" d'intérêt) en vision par ordinateur consiste à essayer de trouver de manière algorithmique des points (ou zones) "intéressantes" d'une image. Par « intéressant » on entend que ces points vont nous permettre de caractériser une image, dit encore différemment qu'ils présentent de propriétés remarquables. Il existe de nombreuses méthodes de détection de points ou de zones d'intérêt et il est hors du propos de ce TP de vous les présenter toutes.
En résumé, un détecteur de point d'intérêt (*feature detector*) va `prendre en entrée une image` et va vous `fournir en sortie un ensemble de coordonnées (pixels)` des points calculés comme étant intéressants par la méthode employée dans le détecteur. Notez donc

qu'un *feature detector* vous fournit **uniquement** les coordonnées des points d'intérêt, mais rien d'autre.
Il existe un grand nombre de *feature detectors* : Harris, FAST, SIFT, Shi-Tomasi, ORB, etc. nous y reviendrons plus tard.

1.2 Description des points d'intérêt

Un descripteur de points d'intérêt (*feature descriptor*) est un algorithme qui a pour objectif de caractériser des points d'intérêt d'une image.
Il va prendre en entrée : `une image et un ensemble de points d'intérêt` et va fournir en sortie `un vecteur de descripteurs pour chacun des points d'intérêt`.

Les *feature descriptors* vont encoder pour chaque point d'intérêt un ensemble d'informations caractéristiques de ce point. Il calcule donc une sorte de signature pour chacun des points d'intérêt qui permet de les différencier.
Ils `ont besoin de l'image en plus de la liste de points d'intérêt car les descripteurs sont très souvent basés sur des informations du voisinage des points d'intérêt` (gradients de couleurs, d'intensité, etc.). Ils encodent aussi très souvent d'une manière ou d'une autre l'échelle associée à un point d'intérêt, en effet, le changement d'échelle d'une image peut fortement impacter la détection de points d'intérêt, et donc leurs descriptions.
Idéalement un *feature descriptor* va posséder un ensemble de bonnes propriétés comme par exemple :

- invariance à la rotation
- invariance à la mise à l'échelle
- ou plus généralement invariance par rapport à une transformation géométrique

et ce afin de pouvoir `identifier un même point d'intérêt même si l'image a subit une transformation géométrique`. En effet, on aimerait pouvoir dire qu'un point d'intérêt est identique dans une image I et dans cette même image I tournée de 45°.
Il existe beaucoup d'algorithmes différents de description de points d'intérêt : SIFT (qui est à la fois un détecteur et un descripteur), SURF, ORB, AKAZE, etc.

1.3 Mise en correspondance des points (Appariement)

La mise en correspondance de *features* revient à la question suivante :
"Étant donné une *feature* dans l'image I_1 comment trouver celle qui lui correspond le mieux dans l'image I_2 ?"
L'idée pour répondre à cette question est la suivante :

1. Définir une fonction de distance permettant de comparer les descripteurs de plusieurs *features*
2. Tester toutes les *features* de I_2 et trouver celle qui a la distance minimale par rapport à la fonction définie ci-dessus.

On peut résumer visuellement l'idée comme en Figure 1.2, source `https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf`.

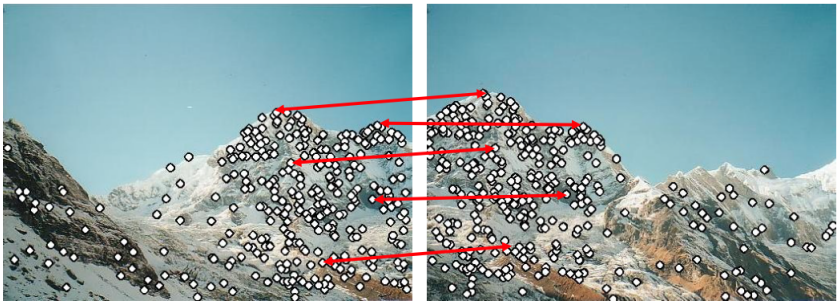

FIGURE 1.2 – Idée de l'appariement de descripteurs visuels.

Il existe plusieurs fonctions pour calculer le matching entre deux ensembles de descripteurs :
- `SSD` : *Sum of Square Differences* entre les deux descripteurs
- Ratio de `SSD`
- Distance `L1`, `L2`, `Hamming`, etc.

Et plusieurs manières de faire le matching :
- `Force Brute` : on teste tous les descripteurs 2 à 2. On est surs d'avoir le meilleur résultat, mais c'est très très lent.
- Basé `FLANN` : algorithme basé voisins les plus proches (FLANN = *Fast Library for Approximate Nearest Neighbors*). On essaye d'être malins et de ne pas tester toutes les combinaisons possibles. Marche beaucoup plus rapidement sur les gros volumes de données.

Enfin, une fois la fonction de distance choisie et la manière de faire le matching, il faut également `choisir un seuil` permettant de dire quand `un matching est bon`.
Il existe encore plusieurs manières de choisir ce seuil, une classique est de se baser sur `la distance minimum calculée pendant le matching`. En général on choisit donc le seuil en multipliant la distance minimum ($distMin$) par un coefficient multiplicateur (α, généralement de l'ordre de 3-20) :

$$seuil = \alpha * distMin \tag{1.1}$$

Nous verrons plus loin comment implémenter cette mise en correspondance en OpenCV.
Pour aller plus loin :
- `https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf`
- `https://www.cs.toronto.edu/~urtasun/courses/CV/lecture04.pdf`

1.4 CALCUL DE LA TRANSFORMATION DE L'IMAGE PAR HOMOGRAPHIE

Enfin, une fois le matching effectué, c'est-à-dire que nous possédons deux ensembles de points qui sont en correspondance, il nous reste à `estimer la transformation géométrique qui relie ces deux ensembles`. Comme nos deux ensembles de points sont coplanaires, cette transformation géométrique sera une `homographie`.
Voici quelques informations sur les homographies :

— https://fr.wikipedia.org/wiki/Application_projective
— http://devernay.free.fr/cours/vision/pdf/vision3_projective.pdf
— https://www.iro.umontreal.ca/~roys/ift6145H06/calib4.pdf
— http://math.univ-lille1.fr/~blanccen/Enseignement/Agreg/Polys/AnalyseComplexe/Homographies.pdf
— http://www.bibmath.net/dico/index.php?action=affiche&quoi=./h/homographie.html
— https://en.wikipedia.org/wiki/Homography_(computer_vision)

L'idée est donc, à partir de deux ensembles de points (les points caractéristiques qui ont été marqués comme "bonnes correspondances") GM_g et GM_d, d'estimer la matrice $3x3$ d'homographie H :

$$H = estimateHomography(GM_G, GM_D) \tag{1.2}$$

Il nous reste donc à appliquer H à l'image de droite, pour ensuite la coller à l'image de gauche (voir Figure 1.1).

2 Implémentation OpenCV

Le but de ce TP sera d'implémenter les concepts évoqués plus haut en OpenCV en utilisant le squelette fourni sur le serveur pédagogique.

2.1 Implémentation d'un *feature detector* en OpenCV

Avant toute chose, sachez que OpenCV fournit un ensemble de tutoriels (en C++ et en Python) plus ou moins détaillés sur les notions abordées dans ce TP :
— https://docs.opencv.org/4.x/d7/d66/tutorial_feature_detection.html
— https://docs.opencv.org/4.x/d5/dde/tutorial_feature_description.html
— https://docs.opencv.org/4.x/d5/d6f/tutorial_feature_flann_matcher.html
— https://docs.opencv.org/4.x/d7/dff/tutorial_feature_homography.html

OpenCV vous fournit une classe "générique" : Feature2D que l'on peut instancier selon différents types de détecteurs implémentés dans OpenCV. Ainsi Feature2D peut être instancié en pour donner un détecteur :
— Shi-Tomasi (Good Features to Track)
— SIFT,
— ORB,
— etc.

Plus d'informations sont données ici :
https://docs.opencv.org/4.5.5/d0/d13/classcv_1_1Feature2D.html

Notez que les deux détecteurs "les plus performants" pour ce que l'on souhaite faire aujourd'hui sont SIFT et SURF. Le détecteur et descripteur SURF n'est pas disponible dans la version Python d'OpenCV (même avec les contrib) car il était jusqu'à il y a peu sous brevet.

Ainsi, nous allons devoir utiliser ceux fournis par OpenCV. Dans ce TP **nous vous conseillons d'utiliser** `ORB` et `SIFT` : en effet, `ORB` et `SIFT` fournissent à la fois un détecteur de points d'intérêt et un descripteur!

QUESTION 1 : PRENEZ CONNAISSANCE DU SQUELETTE FOURNI. Regardez les fichiers fournis afin de comprendre ce que le code fait. Complétez le code donné pour permettre la *détection* et l'*affichage* de points d'intérêt autres que ceux de donnés par SIFT.
Rappel : l'idée du code est de charger 2 images (une "gauche" appelée `img1` et une "droite" appelée `img2`) que l'on va essayer de "recoller". On va détecter des points d'intérêt, les décrire, les mettre en correspondance puis calculer la transformation géométrique qui permet de faire passer un ensemble dans l'autre. Cela nous permet de modifier l'image "droite" et donc de recréer un "panorama".
À faire : Ajoutez dans votre rapport une capture d'écran montrant les résultats dans les deux images. Illustrez avec différentes valeurs de nombre de points d'intérêt détectés.

QUESTION 2 : DESCRIPTION DE POINTS D'INTÉRÊT - *feature detector*. Modifier le squelette fourni pour appeler une fonction qui en plus de détecter les points d'intérêt SIFT ou ORB va créer les descripteurs correspondants avec la fonction *compute*. Vous pouvez vous aider de `https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html` et `https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html`. Attention, le détecteur de Shi-Tomasi ne permet de calculer des descripteurs.

QUESTION 4 : FAIRE LA MISE EN CORRESPONDANCE. Pour calculer les descripteurs en correspondance nous allons faire appel à la méthode de mise en correspondance "brute force" avec différence normes possibles (NORM_L1, NORM_L2, NORM_HAMMING, NORM_HAMMING2) comme décrit dans la documentation
`https://docs.opencv.org/3.4/d3/da1/classcv_1_1BFMatcher.html`
Vous définirez aussi au seuil comme décrit précédemment :

$$seuil = \alpha * minDist \tag{2.1}$$

Ces bonnes correspondances sont à sauvegarder un vecteur `bestMatches` défini pour l'occasion.
Pour faire la fonction qui prend en paramètre le type de mise en correspondance, les descripteurs des deux images, le seuil choisi et qui renvoie en résultat les meilleures correspondances, vous pouvez vous inspirer de `https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html`.

QUESTION 4 : AFFICHER LE RÉSULTAT DE LA MISE EN CORRESPONDANCE. Une fois les meilleures correspondances trouvées, affichez-les entre les deux images avec `cv.drawMatches`. Illustrez ce résultat dans votre rapport.

QUESTION 5 : CALCULER À PARTIR DES POINTS EN CORRESPONDANCE L'HOMOGRAPHIE PERMETTANT DE TRANSFORMER LA 2E IMAGE DANS LE POINT DE VUE DE LA 1RE. Vous pouvez vous inspirer du tutoriel suivant `https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html`

QUESTION 6 : COLLER LES DEUX IMAGES ENSEMBLE POUR FAIRE UN PANORAMIQUE. A partir de l'image de gauche et de l'image de droite qui a été transformée par application d'une homographie, créez une 3e image (du point de vue de la caméra ayant pris l'image 1) avec les deux images recollées. Affichez le résultat dans une nouvelle fenêtre.

QUESTION 7 : CHANGEMENT D'IMAGES ? Illustrez les résultats avec d'autres images de votre choix.

QUESTION 8 : CHANGEMENT DE DÉTECTEUR ET DESCRIPTEUR. Testez de créer un autre type de *feature detector* et d'autres descripteurs et illustrez les résultats obtenus.