ECOLE CENTRALE DE NANTES

Option Réalité virtuelle - TP VSION Image Stitching

13 décembre 2021

1 TP IMAGE STITCHING – POINTS CARACTÉRISTIQUES, DESCRIPTEURS, APPARIEMENT, HOMOGRAPHIE ETC.

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.

2 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 *stitching* d'images (collage, voir Figure 2.1).

Vous verrez qu'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;

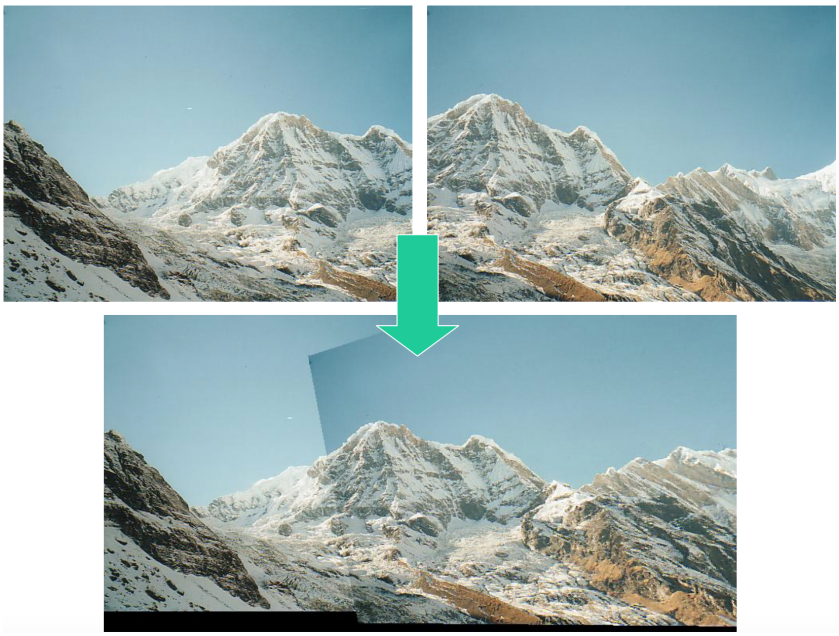

FIGURE 2.1 – Objectif final du TP.

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).
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 C++/OpenCV ainsi que quelques images permettant de réaliser ce collage d'images.

2.1 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.
Pour plus d'informations, nous vous renvoyons au cours et aux ressources suivantes (à regarder après le TP) :

— https://fr.wikipedia.org/wiki/Détection_de_zones_d%27intérêt
— https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)

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.

2.2 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 "*fingerprint*" pour chacun des points d'intérêt qui permet de les différencier.

Les *feature desriptors* 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.

Pour plus d'informations :

— https://fr.wikipedia.org/wiki/Extraction_de_caractéristique_en_vision_par_ordinateur
— https://en.wikipedia.org/wiki/Visual_descriptor
— https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)

— https://fr.wikipedia.org/wiki/Scale-invariant_feature_transform

2.3 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 2.2, source https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf.

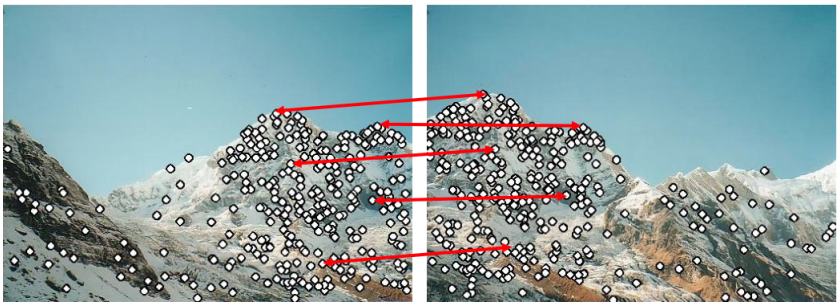

FIGURE 2.2 – Idée de l'appariement de descripteurs visuels.

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) :

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

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

2.4 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 (à lire après le TP) :

— 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 marqué comme "bons matchs") GM_g et GM_d, d'estimer la matrice $3x3$ d'homographie H :

$$H = estimateHomography(GM_G, GM_D) \tag{2.2}$$

Il nous reste donc à appliquer H à l'image de droite, pour ensuite la coller à l'image de gauche (voir Figure 2.1).

3 Implémentation OpenCV

Vous êtes maintenant familiers avec OpenCV. Le but de ce TP sera d'implémenter les concepts évoquées plus haut en OpenCV en utilisant le squelette fourni sur Hippocampus.

3.1 Implémentation d'un *feature detector* en OpenCV

Avant toute chose, sachez que OpenCV fournit un ensemble de tutoriels plus ou moins détaillés sur les notions abordées aujourd'hui. Je vous conseille donc d'aller jeter un œil sur ces ressources une fois le TP terminée pour plus d'informations :

— https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning
— https://docs.opencv.org/3.0-beta/doc/tutorials/features2d/feature_detection/feature_detection.html#feature-detection

Nous vous fournissons pour le `feature detector` la classe C++ `MyFeatureDetector` (et donc les fichiers `MyFeatureDetector.hpp` et `MyFeatureDetector.cpp`).

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 detecteur :

- `Harris`,
- `Shi-Tomasi`
- `SIFT`,
- `SURF`,
- `ORB`,
- etc.

Plus d'informations sont données ici :
`https://docs.opencv.org/3.0.0/d5/d51/group__features2d__main.html`

Notez que les deux détecteurs "les plus performants" pour ce que l'on souhaite faire aujourd'hui sont `SIFT` et `SURF`. Le problème est que ces deux détecteurs sont protégés par des brevets et donc non libres de droits...

Ainsi, nous allons devoir utiliser ceux fournit par OpenCV. Dans ce TP **nous vous conseillons d'utiliser** `ORB` : en effet, `ORB` fournit 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.
Rappel : l'idée du code est de charger 2 images (une "gauche" appelée `im1` et une "droite" appelée `im2`) 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 une ensemble dans l'autre. Cela nous permet de modifier l'image "droite" et donc de recréer un "panorama".

QUESTION 2 : CRÉER UN *feature detector*. Modifiez le squelette fourni (c'est-à-dire les fichiers `MyFeatureDetector.cpp` et `.hpp`) afin de pouvoir créer un *feature detector* `ORB`.
Il "suffit" de créer l'attribut `_myFeatureDetector` en appelant la méthode `create` de la "bonne classe" OpenCV (celle qui correspond au détecteur que vous souhaitez) et avec les bons paramètres.
Attention au(x) paramètre(s) d'entrée de la méthode `create`.

QUESTION 3 : AFFICHER LES POINTS D'INTÉRÊT DÉTECTÉS. Modifiez le fichier `main.cpp` afin d'afficher les points d'intérêt détectés dans les deux images. Indice : regardez la méthode `displayFeatures` :) et les TODOs.
À faire : Ajoutez dans votre rapport une capture d'écran montrant les résultats dans les deux images. Illustrez avec différentes valeurs du premier paramètre du constructeur du détecteur `ORB`, dites aussi à quoi correspond ce paramètre.

QUESTION 4 : INSTANCIER LES DESCRIPTEURS. Dé-commentez dans le fichier `main.cpp` les lignes correspondantes aux deux objets `_myFeatureExtractor1` et `_myFeatureExtractor2`. OpenCV nous fournit une classe générique `DescriptorExtractor` dans le même ordre d'idée. Modifiez les fichiers `MyDescriptorExtractor.cpp` (et `.hpp`) afin de créer deux descripteurs `ORB`. Là encore, il "suffit" d'appeler le bon constructeur.

QUESTION 5 : CALCULER LES DESCRIPTEURS. Complétez les fichiers `main.cpp` et `MyDescriptorExtractor.cpp` afin de calculer les descripteurs des deux ensembles de points d'intérêts détectés plus haut. Le code est à mettre dans la méthode `computeDescriptors` et il vous faudra faire les "bons" appels dans la méthode `main`.

QUESTION 6 : FAIRE LA MISE EN CORRESPONDANCES. Complétez le code de la méthode `match` de la classe `MyDescriptorMatcher`. L'idée est de garder **uniquement** les mises en correspondance qui sont d'une distance inférieure à un seuil. Le seuil est fixé à :

$$seuil = \alpha * minDist \tag{3.1}$$

Ces bons matchs sont à sauvegarder dans le vecteur `bestMatches` définis pour l'occasion. Vous ne devez pas supprimer du vecteur qui contient tous les matches.

QUESTION 7 : AFFICHER LE RÉSULTAT DE LA MISE EN CORRESPONDANCE. En appelant la méthode `drawMatchingResults` de la classe `MyDescriptorMatcher`, modifiez le code de la méthode `main` pour afficher le résultat du matching. Illustrez ce résultat dans votre rapport.

QUESTION 8 : COMPRENDRE L'EXTRACTION DES MEILLEURS RÉSULTATS DE LA MISE EN CORRESPONDANCE POUR CALCULER L'HOMOGRAPHIE. Expliquez ce que fait le code de la méthode `main` compris entre les commentaires :
`COMMENT CODE BELOW` et `COMMENT CODE ABOVE`

QUESTION 9 : COMPRENDRE LE CALCUL DE L'HOMOGRAPHIE ET DU RÉSULTAT DU "STITCHING". Expliquez ce que fait le code de la méthode `main` compris entre les commentaires :
`BELOW: FIND THE HOMOGRAPHY MATRIX AND STICH IMAGE` et
`ABOVE: FIND THE HOMOGRAPHY MATRIX AND STICH IMAGE`

QUESTION 10 : ILLUSTREZ LE RÉSULTAT OBTENU. Illustrez le résultat obtenu avec les deux images fournies et chargées par défaut.

QUESTION 11 : CHANGEMENT D'IMAGES ? Illustrez les résultats avec d'autres images de votre choix.

QUESTION 12 : CHANGEMENT DE DÉTECTEUR ET DESCRIPTEUR. Testez de créer un autre type de *feature detector* et d'autres descripteurs et illustrez les résultats obtenus.