Devoir #2

Panorama

INF889G - Vision par ordinateur

Hiver 2024

Table des matières

Création d'un panorama

Ce devoir couvre beaucoup de sujets : la recherche des points caractéristiques dans une image; la description de ces points caractéristiques; le pairage de ces points dans une autre image; le calcul de la transformation d'une image vers une autre; et l'assemblage des images en un panorama.

L'algorithme de haut niveau est déjà réalisé pour vous ! Vous pouvez le trouver près du bas du fichier `panorama_image.py`, et ressemble approximativement à :

```
def panorama_image(im_a, im_b, sigma, thresh, nms, inlier_thresh, iters, cutoff):
    """Création d'un panorama à partir de deux images"""
    # Calcul des coins et des descripteurs
    ad = harris_corner_detector(im_a, sigma, thresh, nms, an)
    bd = harris_corner_detector(im_b, sigma, thresh, nms, bn)

    # Détection des correspondances
    m = match_descriptors(ad, an, bd, bn, mn)

    # Exécuter RANSAC pour trouver l'homographie
    H = RANSAC(m, mn, inlier_thresh, iters, cutoff)

    # Assemblage des images ensemble avec l'homographie
    panorama = combine_images(a, b, H)
    return panorama
```

Pour résumer, nous trouverons d'abord les coins dans chaque image à l'aide d'un détecteur de coin de Harris. Ensuite, nous trouverons les correspondances entre les descripteurs pour chaque coin. Nous utiliserons RANSAC pour estimer la projection entre le système de coordonnées d'une image vers celui de l'autre image. Finalement, nous assemblerons les images ensemble en utilisant cette projection.

> **Note** : La majorité du travail pour ce devoir consiste à modifier les fichiers `harris_image.py` et `panorama_image.py`. Utilisez un notebook nommé `devoir2.ipynb` pour tester votre code, pour afficher des résultats, et pour ajouter des commentaires au besoin.

Détection des coins de Harris

Nous allons implémenter la détection des coins Harris comme discuté en classe. L'algorithme de base est :

1. Calcul des dérivées partielles de l'image I_x et I_y.
2. Calcul des métriques I_xI_x, I_xI_y, et I_yI_y.
3. Calcul des composantes du tenseur de structure en tant que somme pondérée des métriques du voisinage.
4. Calcul de la similarité à un coin de Harris (*Harris «cornerness»*) en estimant la 2^e valeur propre : $\det(S) - \alpha\text{trace}(S)^2$, $\alpha = 0.06$.

5. Exécuter la suppression des non-maximums sur la réponse précédente.

Calcul du tenseur de structure

Complétez la fonction `structure_matrix(im,sigma)` dans le fichier `harris_image.py`. Ceci effectuera les 3 premières étapes de l'algorithme : calcul des dérivées, des métriques correspondantes, et la somme pondérée des informations des dérivées pour un voisinage. Comme discuté en classe, la somme pondérée peut être facilement calculée avec un lissage gaussien.

Calcul de la similarité à un coin

Complétez la fonction `cornerness_response(S)` dans le fichier `harris_image.py`.

Suppression des non-maximums

On veut seulement les réponses locales maximums de notre détecteur de coin afin de faciliter le pairage des points caractéristiques. Complétez la fonction `nms_image(im, w)` dans le fichier `harris_image.py`.

Pour chaque pixel dans l'image `im`, vérifiez chaque pixel dans un voisinage de `w` pixels (Distance de Chebyshev). D'une façon alternative, vérifiez une fenêtre de taille $2w + 1$ centrée sur chaque pixel. Si au moins une valeur dans le voisinage est plus élevée, supprimez la valeur du pixel central (assignez-lui une valeur négative très basse).

Compléter le détecteur de Harris

Complétez les sections manquantes de la fonction `harris_corner_detector(im, sigma, thresh, nms)`. Cette fonction devrait retourner un tableau de descripteurs pour les coins dans l'image. Le code pour calculer les descripteurs est fourni.

Après avoir complété cette fonction, vous devriez être en mesure decalculer les coins et leur descripteur pour une image ! Essayez d'exécuter :

```
import imageio
import matplotlib.pyplot as plt
from harris_corner import detect_and_draw_corners

im = imageio.imread("data/Rainier1.png")
im_cd = detect_and_draw_corners(im, 2, 0.0004, 3)
plt.imshow(im_cd); plt.show()
```

Ceci détectera les coins en utilisant une fenêtre gaussienne de `sigma=2`, un seuil de "*cornerness*" de 0.0004, et une distance nms de 3 (ou une fenêtre de taille $7x7$). Vous devriez obtenir un résultat similaire à ceci :

Les coins sont représentés par des croix. L'algorithme donne des résultats sensés, beaucoup de coins sont près de l'endroit où la neige rencontre la roche. Essayez de jouer avec les différentes valeurs des paramètres pour voir comment ils affectent votre détecteur de coin.

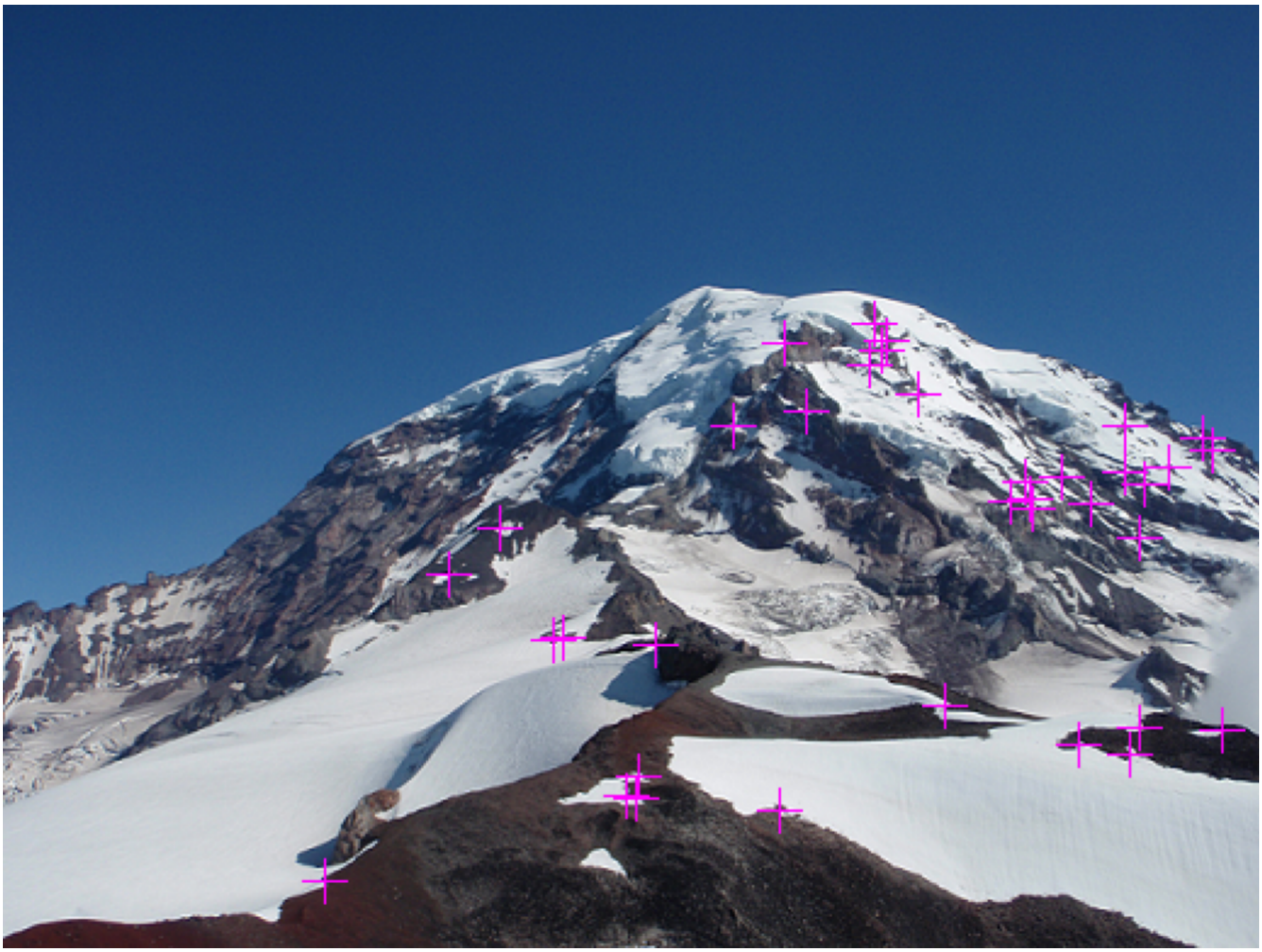

Figure 1: Exemple de résultat pour la détection des coins de Harris

Correspondance des patchs

Pour obtenir un panorama, nous devons faire correspondre les coins détectés dans une image avec les coins correspondant dans l'autre image. Le code du descripteur est déjà écrit pour vous. Le descripteur se compose de l'intensité des pixels voisins, mais la valeur du pixel central a été soustraite. Cela nous donne une petite quantité d'invariance aux conditions d'éclairage.

Pour la suite du devoir, modifiez le fichier `panorama_image.py`

Métrique de distance

Pour comparer les patchs, nous utiliserons la distance L1. L'erreur quadratique (distance L2) peut entraîner des problèmes avec les valeurs aberrantes. Nous ne voulons pas que quelques pixels aberrants perturbent la fonction de correspondance. La distance L1 (somme des différences absolues) se comporte mieux avec certaines valeurs aberrantes.

Implémentez la fonction `l1_distance(a,b,n)` permettant de calculer la distance entre deux vecteurs ou deux nombres réels (`float`). Les vecteurs `a`, `b` et le nombre de valeurs qu'ils contiennent sont fournis en argument.

Pairage des descripteurs

Détection du meilleur pairage de `a` vers `b`

Nous allons d'abord examiner les descripteurs de l'image `a` et trouver la meilleure correspondance avec les descripteurs de l'image `b`. Complétez le premier `TODO` dans la fonction `match_descriptors(a,b)`.

Éliminer les patchs multiples pour un même descripteur de `b`

Chaque descripteur dans l'image `a` apparaîtra dans un seul pairage. Par contre, plusieurs d'entre eux pourraient correspondre avec le même descripteur dans l,image `b`. Ceci peut être problématique. Par exemple, si un ensemble de correspondances va au même point, il existe une homographie facile à estimer qui réduit simplement l'image entière à un point pour se projeter de `a` à `b`. Mais nous savons que c'est faux. Alors, débarrassons-nous simplement de ces correspondances en double et faisons en sorte que nos correspondances soient individuelles (1 descripteur `a` pour 1 descripteur `b`).

Pour ce faire, triez les correspondances en fonction de leur distance afin que la distance la plus courte soit la première. Un comparateur a est fourni que vous pouvez utiliser, recherchez simplement comment appliquer correctement la méthode `sort` si vous ne le savez pas déjà. Ensuite, parcourez les correspondances dans l'ordre et gardez une trace des éléments de `b` que nous avons vus. Si on en voit un pour une deuxième (ou une troisième, etc.) fois, on l'ignore, sinon on l'ajoute à une liste `filtered_matches` ! Cela peut être fait en un seul passage dans les données.

Une fois ce filtrage effectué, on peut afficher les correspondances détectées entre les images :

```
import imageio
import matplotlib.pyplot as plt
from panorama_image import detect_and_draw_corners
a = imageio.imread("data/Rainier1.png")
b = imageio.imread("data/Rainier2.png")
m = find_and_draw_matches(a,b,2,25,3)
plt.imshow(m); plt.show()
```

Vous devriez obtenir un résultat similaire à celui-ci.

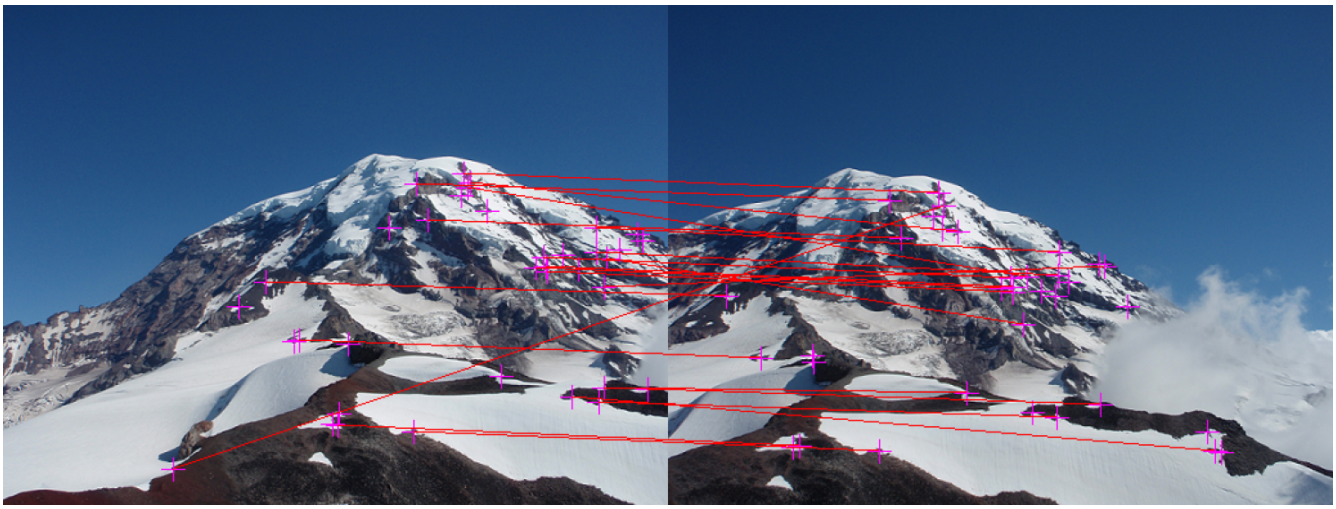

Figure 2: Exemple de résultat pour le pairage de caractéristiques

Ajuster la projection aux données

Maintenant que nous avons quelques correspondances, on doit prédire la projection entre les deux ensembles de points. Toutefois, ceci peut être difficile à réaliser, car il y a beaucoup de correspondances bruitées (ex. : faux positifs). Plusieurs correspondances sont correctes, mais il y a quelques correspondances aberrantes dans nos données.

Projection des points par homographie

Implémentez la fonction `project_point(H,p)` pour projeter un point `p` en utilisant la matrice `H`. N'oubliez pas d'effectuer la bonne normalisation pour convertir les coordonnées homogènes vers les coordonnées images après la tranformation.

Calcul de la distance entre 2 points

Complétez la fonction `point_distance(p,q)`. Utiliser la distance L2.

Calcul les *inliers* d'un modèle

Déterminez combien de correspondance sont des *inliers* d'un modèle. Complétez la fonction `model_inliers(H, m, thresh)` pour parcourir chaque point, les projeter en utilisant l'homographie `H`, et vérifier si le point projeté est à au plus une distance `thresh` du point correspondant actuel dans l'autre image.

Ajuster l'homographie

Nous devons résoudre notre problème pour trouver l'homographie en utilisant des opérations matricielles discutées en classe pour résoudre une équation du type $M\vec{a} = \vec{b}$. La majorité de ces opérations est déjà implémentée, vous devez vous occuper de remplir les matrices M et b avec des informations sur les pairages dans le premier `#TODO` de la fonction `compute_homography(matches, n)`.

Vous devez aussi lire le résultat final et remplir la matrice d'homographie.

Correspondance aléatoire

Une des étapes de la méthode RANSAC est de choisir des correspondances au hasard pour estimer un nouveau modèle. Une façon de réaliser est de mélanger (*shuffle*) le tableau contenant les correspondances et de choisir les `n` premiers éléments pour ajuster un modèle.

Implémentez-le mélange de Fisher-Yates dans la fonction `randomize_matches(m,n)`.

Implémentez RANSAC

Implémentez l'algorithme RANSAC dans la fonction `RANSAC(matches, thresh, k, cutoff)`. Le pseudocode de cet algorithme est fourni dans les diapositives du cours.

Combiner les images avec l'homographie

La dernière étape à réaliser est l'assemblage des images ensemble en utilisant l'homographie détectée ! Étant donné deux images et une homographie, assemblez-les avec la fonction `combine_images(a,b,H)`.

Plusieurs des étapes d'assemblage sont déjà complétées. La première étape est de calculer les limites de l'image. Pour faire cela, on doit projeter les coins de l'image `b` dans le système de coordonnées de l'image `a` en utilisant `Hinv`. Ensuite, on peut calcul les «nouvelles» coordonnées et la position de l'image `a` dans le canevas. Collez l'image `a` à l'endroit approprié dans le canevas.

Ensuite, on doit parcourir chaque pixel qui pourrait apparaître dans l'image `b`, calculer le mapping pour vérifier s'il sont dans l'image `b` et si oui, assigner la couleur du pixel avec la couleur appropriée dans l'image `b`. Le mapping pourrait se trouver entre des pixels de l'image `b`, utilisez l'interpolation PPV (*nearest neighbours*) pour calculer la valeur du pixel à cet endroit.

Si tout fonctionne bien, vous devriez maintenant pouvoir créer des panoramas de base :

```
import imageio
import matplotlib.pyplot as plt
from panorama_image import panorama_image

im1 = imageio.imread("data/Rainier1.png")
im2 = imageio.imread("data/Rainier2.png")
pan = panorama_image(im1, im2)
plt.imshow(pan); plt.show()
```

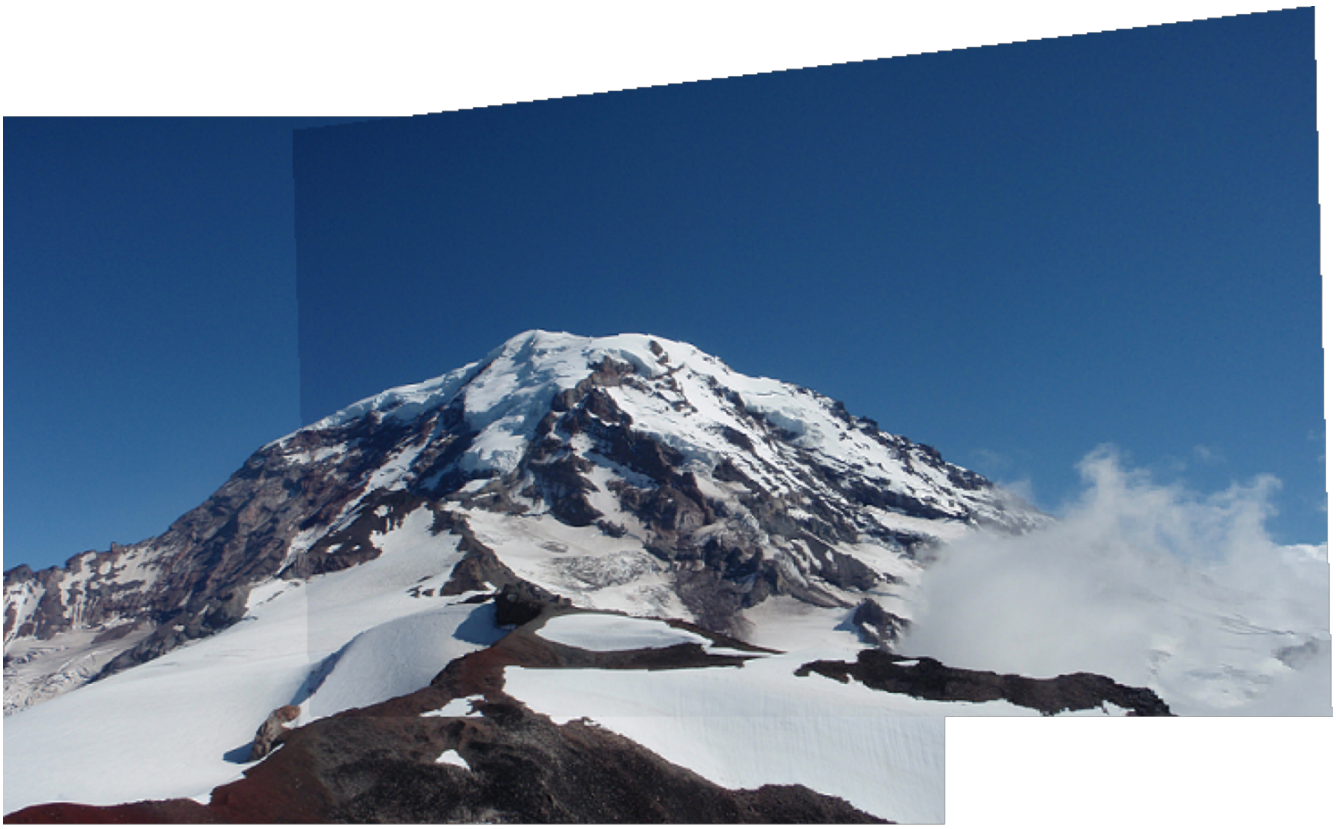

Figure 3: Exemple de panorama

Remise

Remettez sur Moodle vos scripts `harris_image.py` et `panorama_image.py` complétés, et quelques bons panoramas générés avec ceux-ci.

Référence

- Ce devoir a été adapté du devoir 3 du cours CSE455 à l'Université de Washington.