École Polytechnique de Montréal
Département de génie informatique et de génie logiciel

INF8725
Traitement de signaux et d'images
Groupe 1

Construction de panorama par recalage
Points remarquables, Descripteurs et Homographie

Présenté à
Clément Playout
Gabriel Lepetit-Aimon

Soumis par
Jérémie Huppé (1854753)
Antoine Daigneault-Demers (1879075)

4 décembre 2019

Notes sur notre code

Notre application débute dans le fichier app.py (nous utilisons Python 3). L'ensemble de notre code a été commenté afin d'en faciliter la lecture.

Nous avions initialement une implémentation différente de la fonction detection_points_cles (voir fichier detection_points_cles_v1.py). Cette implémentation trouvait les extremas en premier et filtrait ensuite les points de faible contraste et les points sur les arêtes. De plus, les calculs pour trouver les dérivées pour évaluer les points de faible contraste et les points sur les arêtes étaient effectués avec des boucles for. Dans le but d'optimiser notre implémentation, nous avons décidé d'implémenter le calcul de ces dérivés par convolution à l'aide de filtres. De plus, nous avons décidé d'identifier en premier lieu les points de faibles contrastes et les points n'étant pas sur les arêtes et ensuite d'identifier les extremas parmis ces points.

Après l'implémentation de cette nouvelle version de detection_points_cles nous avons observé des comportements étranges par rapport au matching des points-clés selon le nombre d'octaves générées pour identifier les points-clés. Avec un grand nombre d'octaves dans la pyramide de gaussienne nous nous retrouvions avec des couples de points-clés match mauvais. Ce problème est probablement lié au problème décrit à la *Question 2* de la *Section 2.2* qui spécifie que nous avons un nombre de points-clés détectés qui augmente d'octave en octave. Après investigation, nous avons constaté que notre ancienne implémentation detection_points_cles_v1 avait aussi le même problème. Nous n'avons pas été en mesure d'identifier la source de ce problème.

Commentaire à propos de la sélection de l'image correspondant au sigma du point-clé lors de la description de point-clé:

Initialement nous ne savions pas quelle image lissée sélectionner. Nous avions opté pour l'image lissée de taille réduite correspondant au sigma du point-clé. Après avoir vérifié avec Clément, ce dernier nous a affirmé que nous devions prendre l'image de taille originale lissée au sigma du point-clé, ce qui était en effet plus logique. Une fois ce code ajouté au reste de notre implémentation nous avons pu constater d'étranges résultats par rapport aux couples de points-clés. Nous avons donc investigué la situation afin de trouver l'origine de ce changement de comportement lors du matching des points-clés. Après une longue investigation nous n'avons su mettre la main sur le problème. Nous avons finalement décidé de revenir à notre implémentation initiale malgré le fait que nous sommes conscients que cette implémentation ne correspond pas à celle de l'article.

Après une longue investigation afin de trouver la source de ces comportements étranges, nous avons conclu que nous avions probablement un problème dans l'implémentation de notre fonction detection_points_cles ou de notre fonction description_points_cles.

Somme toute, notre génération de panorama fonctionne, mais nous avons jugé important de mentionner ces quelques comportements qui nous semblaient imparfaits.

Section 2

2.1 Construction des différences de Gaussiennes – Barème : 4 points

Question 1

Discutez la relation entre la résolution d'une image et la taille d'un filtre. En particulier, pour un filtre Gaussien fixé, quelle différence existera-t-il entre la convolution avec une image de résolution 1000x1000 et la convolution avec la même image mais à la résolution 500x500 ?

Pour un filtre donné, celui-ci affectera visiblement plus les images à basse résolution que les images à plus haute résolution. Dans le cas d'un filtre Gaussien fixé, la quantité résultante de flou dépendra de la résolution de l'image. En effet, une image de basse résolution sera plus affectée par le filtre, c'est-à-dire qu'elle deviendra plus floue, que la même image à plus haute résolution. Donc, dans le cas discuté, l'image avec la résolution 500x500 deviendra plus floue que celle 1000x1000.

Question 2

Il est possible d'optimiser la construction de la pyramide afin d'éviter de construire des filtres Gaussien excessivement larges (ce qui ralentirait les opérations de convolutions). Pour cela, nous donnons la propriété suivante :

$$\mathbf{G}_{\sigma_2}(\mathbf{G}_{\sigma_1}(\mathbf{u})) = \mathbf{G}_{\sqrt{\sigma_1^2+\sigma_2^2}}(\mathbf{u})$$

où $G\sigma_1(u)$ *représente l'opération de convolution de l'image u avec une gaussienne de taille s1 En vous appuyant sur cette relation, expliciter une relation de récurrence permettant d'exploiter cette propriété pour limiter la taille des filtres Gaussiens lors de la construction de l'échelle de Gaussienne.*

On pose σ_3 la taille du filtre qu'on souhaite comme résultat final. À l'aide de la relation présentée ci-dessus, il est possible de développer l'équation afin d'obtenir σ_2 qui représente la taille du filtre qu'on doit appliquer à l'image qui a déjà été filtrée avec un filtre de taille σ_1.

$$\sigma_3 = \sqrt{\sigma_1^2 + \sigma_2^2}$$

$$\sigma_3^2 = \sigma_1^2 + \sigma_2^2$$

$$\sigma_2^2 = \sigma_3^2 - \sigma_1^2$$

$$\sigma_2 = \sqrt{\sigma_3^2 - \sigma_1^2}$$ C.Q.F.D.

Donc, appliquer le filtre de taille σ_2 à l'image préalablement filtrée avec un filtre de taille σ_1 permet d'obtenir le même résultat que si nous avions appliqué le filtre de taille σ_3 à l'image originale.

Question 3

Tracez la pyramide de gaussienne construite à l'octave de votre choix. Vous utiliserez au choix l'une des images fournies en ressources. (Voir 2.1) Indiquez pour chaque Gaussienne l'échelle correspondante.

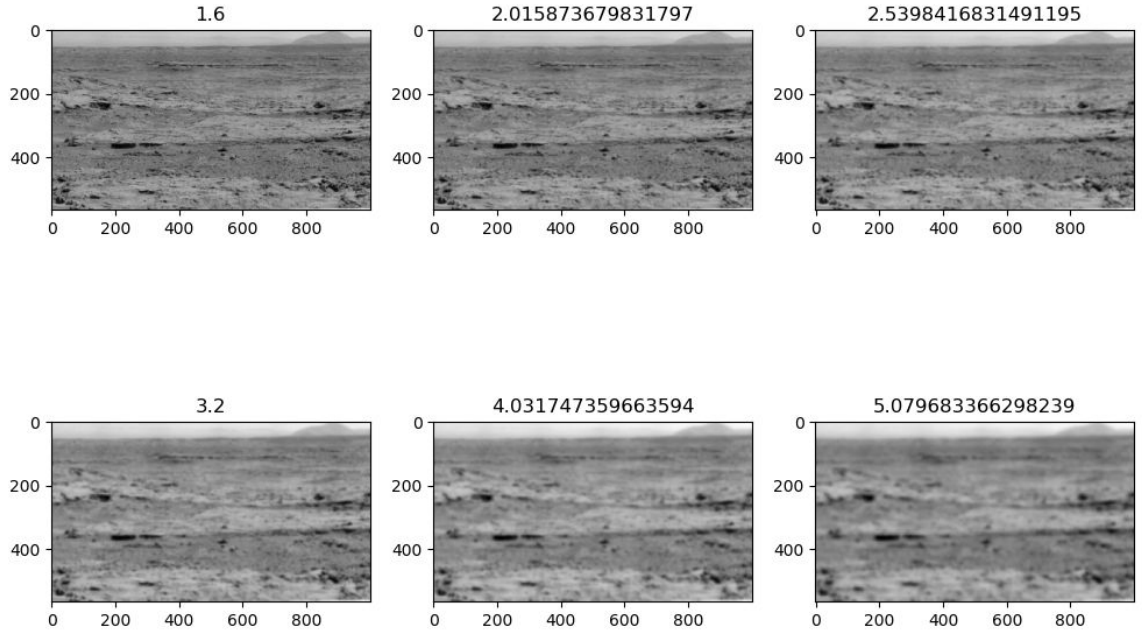

Figure 1: Images pyramide de gaussiennes pour l'octave 1 de l'image droite ayant comme légende leur sigma correspondant.

Question 4

Tracez la différence de Gaussiennes correspondante. Quel type de filtre classique cela vous rappelle-t-il ? (Voir 2.2)

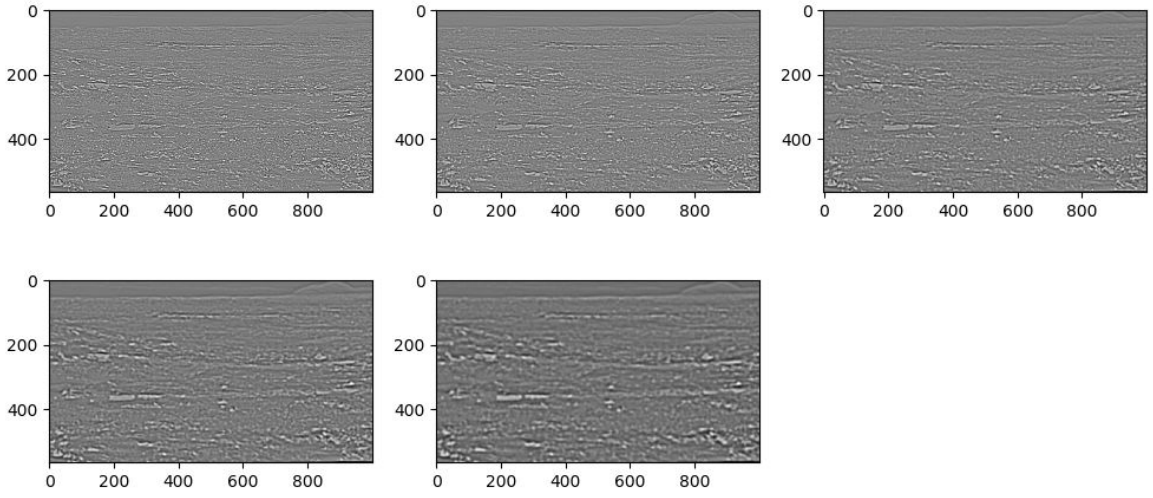

Figure 2: Différences de gaussienne pour l'octave 1 de l'image droite.

Les différences de gaussiennes rappellent les filtres passe-haut qui accentuent les contours.

2.2 Détection des points-clés – Barème : 6 points

Question bonus

Dans l'article de référence, les auteurs suggèrent de corriger la position d'un extremum trouvé en faisant un développement de Taylor à l'ordre 2 sur un espace tridimensionnelle (le point extremum correspondant à un lieu de dérivée nulle). Cette interpolation est une manière de corriger le biais de position de chaque extremum. L'implémentation de cette interpolation n'est pas demandée, vous pouvez donc simplement sauter cette étape. En revanche, tout pseudo-code et/ou code proposant une implémentation de la méthode sera examiné et pourra donner lieu à un point bonus. Le contraste sera donc évalué au point D(x) et non D(ˆ x)

Le pseudo-code se trouve dans le fichier *pseudo_code_obtenir_x_chapeau.py*.

Question 1

Comment calculer de façon efficace (donc sans boucle for) la Hessienne pour tous les pixels de l'image ?

Afin de calculer de façon efficace la Hessienne, il est possible de calculer chaque gradient la constitution, soit dxx, dyy et dxy, par convolution. Voici le détail des filtres à convoluer avec l'image afin d'obtenir respectivement dxx, dyy et dxy:

```
gxx_kernel = np.array([[0., 1., 0.],
                       [0., -2., 0.],
                       [0., 1., 0.],])

gyy_kernel = np.array([[0., 0., 0.],
                       [1., -2., 1.],
                       [0., 0., 0.],])

gxy_kernel = 0.25 * np.array([[1., 0., -1.],
                              [0., 0., 0.],
                              [-1., 0., 1.],])
```

Question 2

Créez trois compteurs comptabilisant respectivement le nombre total d'extrema détectés, le nombre de point de faibles contrastes éliminés et le nombre de points d'arêtes éliminés. Reportez les valeurs obtenues pour chaque octave dans un tableau.

Afin de détecter les extremas, nous commençons par éliminer les points de l'image qui ont un contraste trop faible ou qui font partie d'une arête. Les extremums sont ensuite détectés parmis les points restants.

Notre algorithme parcourt l'image sur 3 échelles à chaque octave. Chaque pixel peut donc être éliminé 3 fois, ce qui explique les nombres de points éliminés parfois plus élevés que le nombre de pixels dans l'image.

Tableau 1: Nombre de points-clés éliminés pour chaque étape de la détection de points-clés.

	Image gauche			Image droite		
	Octave 1	Octave 2	Octave3	Octave 1	Octave 2	Octave 3
Faibles contrastes éliminés	156396	43445	14844	186720	52038	17923
Points d'arêtes éliminés	1036762	243504	49835	1019911	239488	48587
Extremas détectés	637	792	1277	581	719	1202

Ces résultats nous semblent erronés. En effet, nous pensons qu'il serait plus logique d'avoir de moins en moins de points-clés détectés d'octave en octave puisque la taille de l'image diminue considérablement à chaque octave. Nous pensons que cela est peut-être dû à une erreur d'implémentation de notre fonction detection_points_cles ou de notre fonction description_points_cles comme mentionné plus haut dans la section *Notes sur notre code*. Cette explication est valable aussi pour la *Question 3* ci-dessous. En effet, nous nous serions attendus à avoir une évolution décroissante et non croissante du nombre de points-clés détectés en fonction du numéro de l'octave dans l'image 1 et dans l'image 2.

Question 3

Tracez l'évolution du nombre de points conservés ki (keypoints) dans l'octave i en fonction de la résolution de l'octave. R Pour chaque point, vous devriez obtenir un vecteur de dimension 4 de la forme : [#ligne, #colonne, échelle, angle]. En posant Nimg = åi ki, stockez vos points caractéristiques dans une matrice de taille Nimg x 4

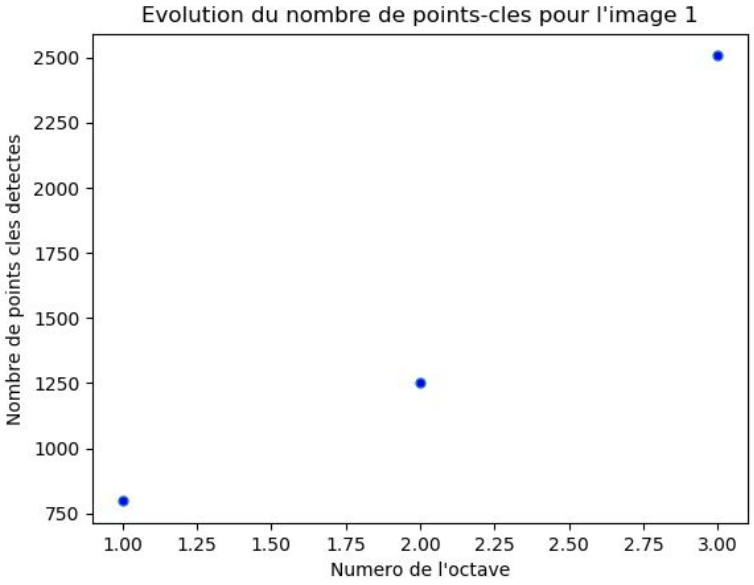

Figure 3: Nombres de points-clés détectés en fonction du numéro de l'octave pour l'image gauche.

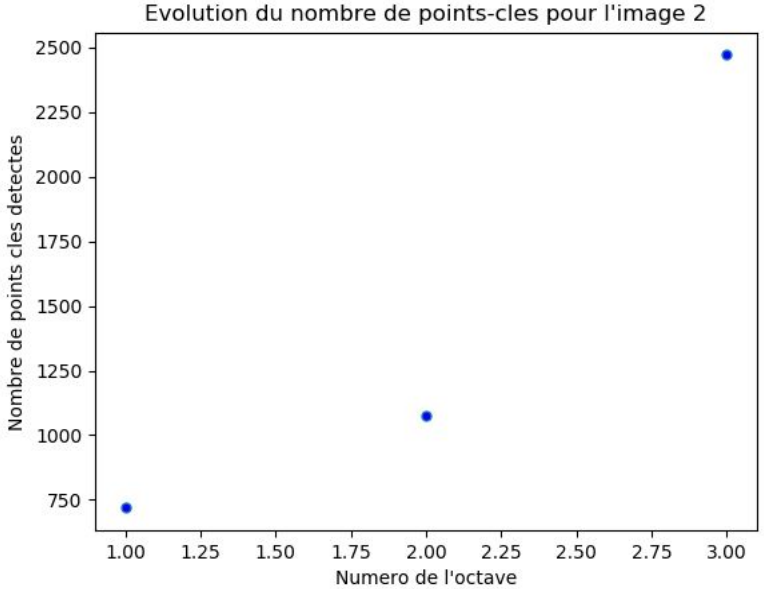

Figure 4: Nombres de points-clés détectés en fonction du numéro de l'octave pour l'image droite.

Les matrices sont sauvegardées dans les fichiers *points_cles_image_droite.npy* et *points_cles_image_gauche.npy*.

Question 4

Pour chacune des images, joignez la matrice obtenue au .zip que vous rendrez. Format : .mat (Matlab) ou .npy (Python)

Les matrices contenant les points-clés trouvés pour chaque image sont enregistrées sous les noms *points_cles_image_gauche.npy* et *points_cles_image_droite.npy*.

Question 5

Sur l'image de votre choix (parmi celles fournies), tracez les lieux de chaque point clé, tel que montré sur la figure 2.3.lic

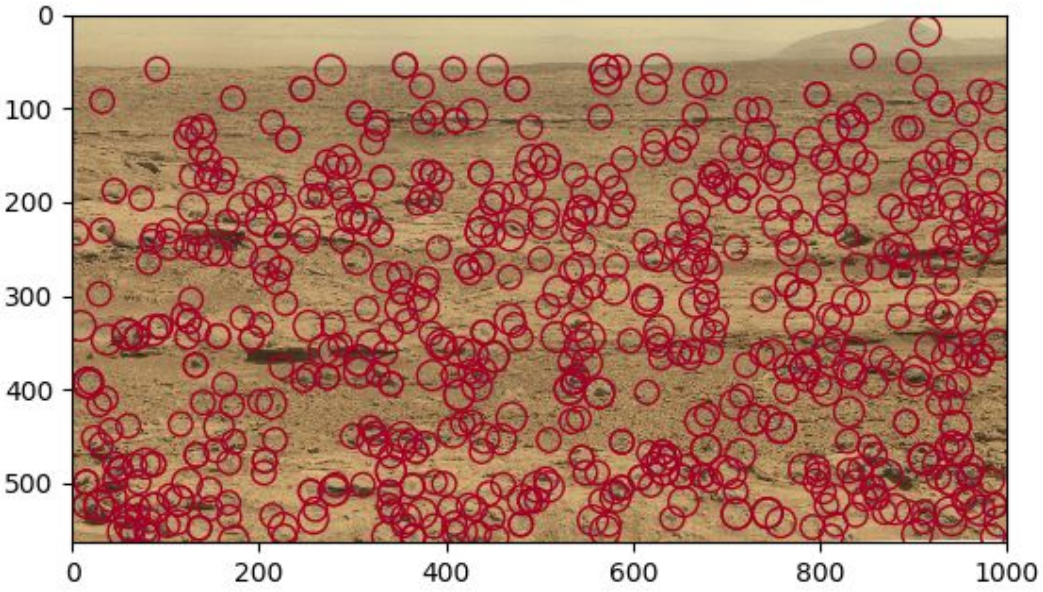

Figure 5.1: Image droite avec points-clés pour 1 octave.

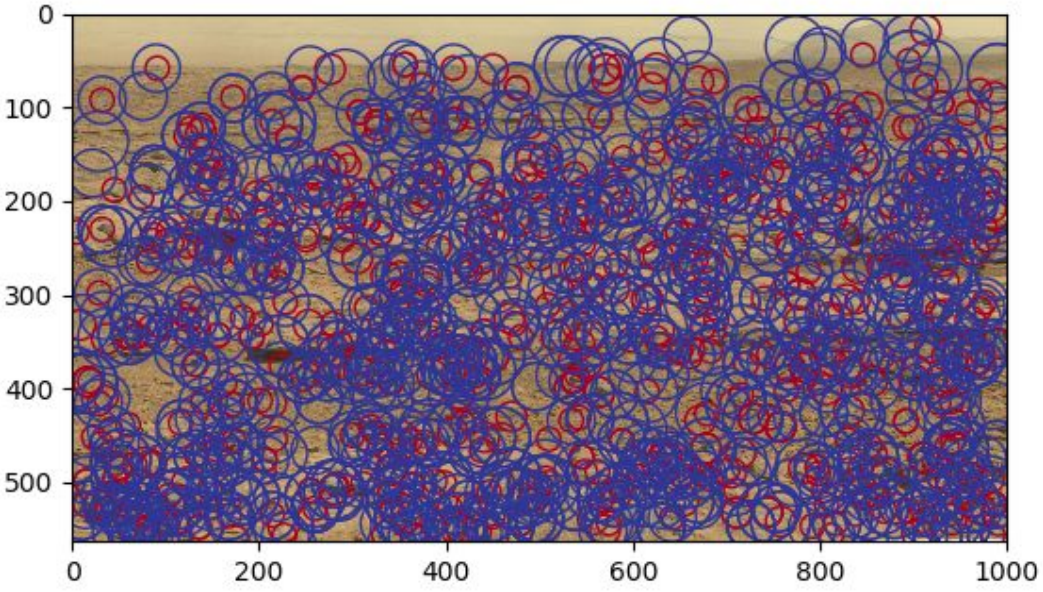

Figure 5.2: Image droite avec points-clés pour 2 octaves.

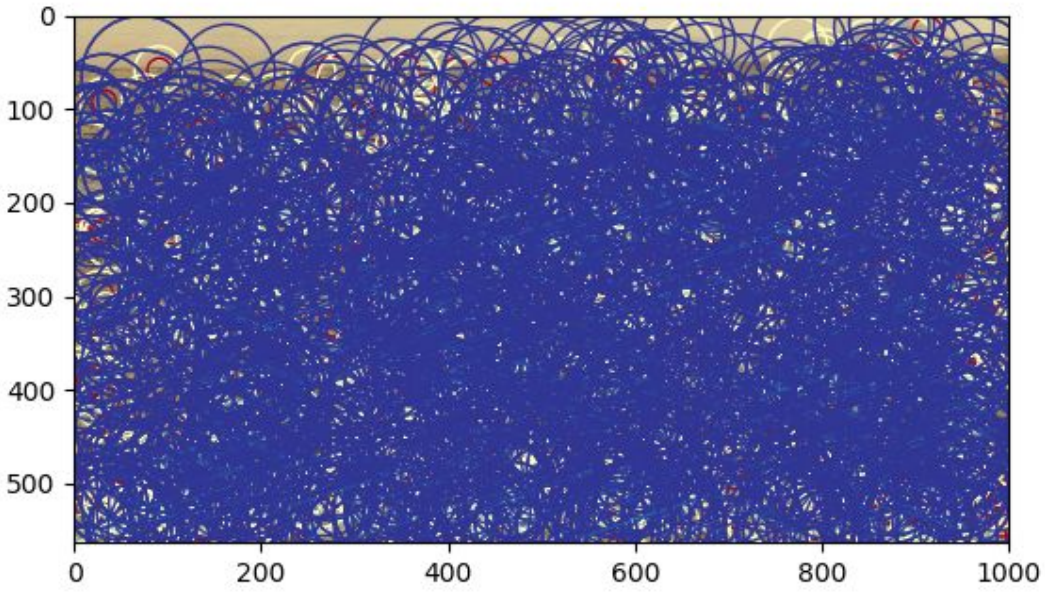

Figure 5.3: Image droite avec points-clés pour 3 octaves.

Note: Des images de lenna avec les points-clés détectés pour 1, 2 et 3 octaves sont disponibles dans le dossier *images_rapport* afin de mieux visualiser l'orientation des points-clés.

2.2.1 Descripteur – Barème : 5 points

Question 1
Rappelez en quelques mots le rôle du vecteur descripteur.

Le vecteur descripteur contient de l'information sur le gradient de 16 sous-régions de dimensions 4x4 autour du point-clé. Cette information permet de distinguer les points-clés de façon unique et robuste. Le vecteur est aussi normalisé pour réduire les effets de la lumière.

Question 2
Avec les valeurs données dans l'article, à chaque point clé, vous devriez obtenir un descripteur de taille 128. Créez un vecteur de 130 éléments. Les deux premiers éléments seront réservés à la position du point clé [#ligne, #colonne]. Les 128 restants seront alloués au descripteur. Pour chaque point, vous obtenez ainsi un vecteur [#ligne, #colonne, descripteur]. Stockez chacun des vecteurs dans une matrice de taille Nimgx130. Pour chacune des images, joignez la matrice obtenue au .zip que vous rendrez. Format : .mat (Matlab) ou .npy (Python)

Les vecteurs descripteurs de l'image droite et de l'image gauche se trouvent respectivement dans les fichiers descripteurs_image_droite.npy et descripteurs_image_gauche.npy

Section 3

3.0.1 Recherche de couples amis – Barème : 2 points

Question 1

Pour chaque descripteur de l'image 1, calculez sa distance euclidienne avec chacun des descripteurs de l'image 2. Tracez la matrice D.

La matrice a été enregistrée sous matrice_D.npy.

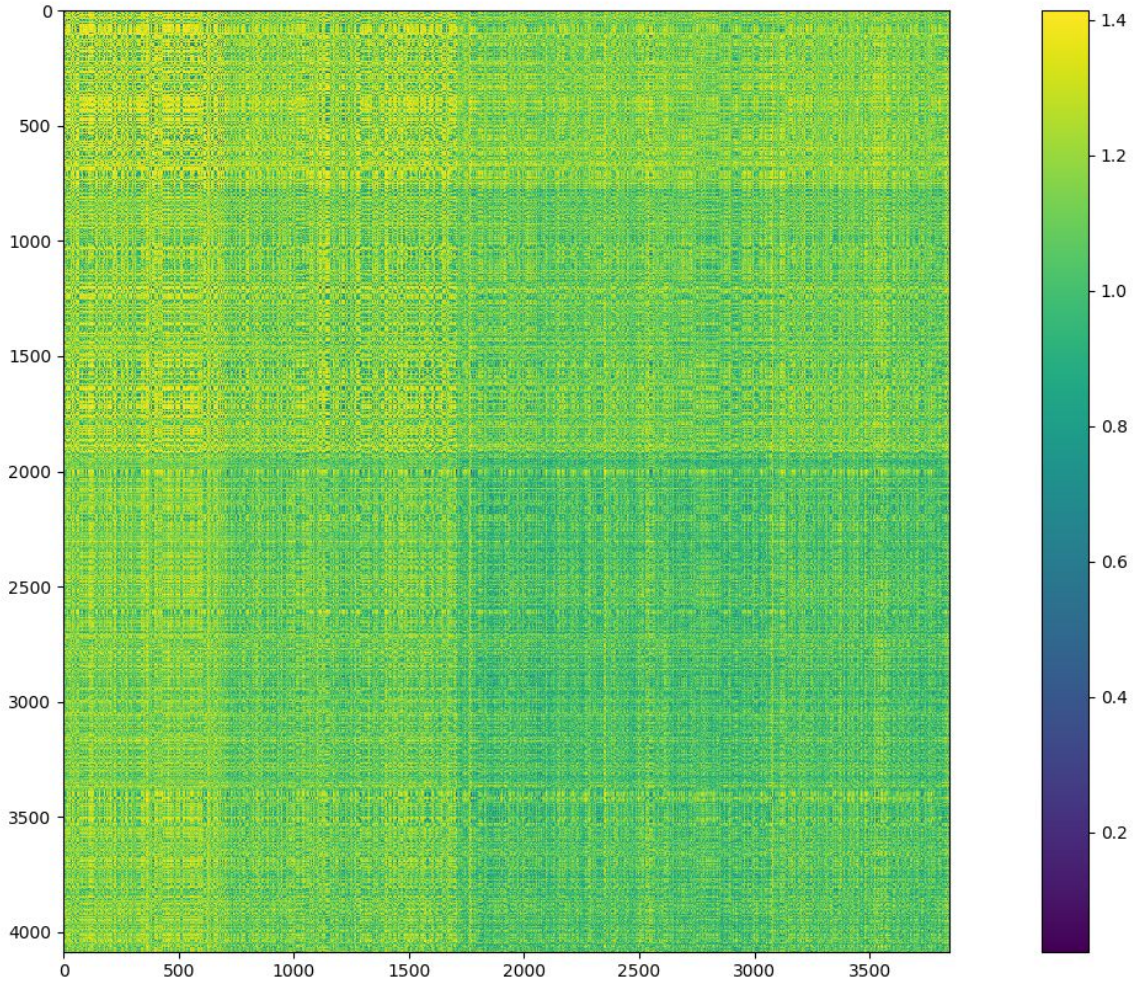

Figure 6: Matrice de distances D.

Question 2

Récupérez les n couples de points de distance minimales (c'est à dire les n plus petites distances de la matrice). Sur chaque image, tracez le lieu des points ainsi obtenus. Vérifiez visuellement que vous obtenez une correspondance entre les deux images.

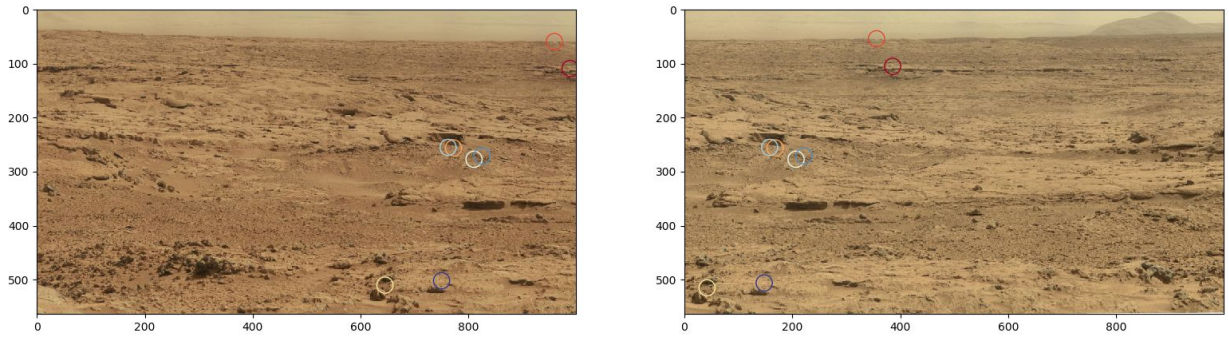

Figure 7: Image de gauche et image de droite avec points-clés correspondants. Chaque couleur correspond à un couple de point.

Il est possible de constater que les points-clés correspondent.

3.0.2 Homographie – Barème : 3 points

Question 1

Vérifiez numériquement que le dernier vecteur singulier de la décomposition SVD est bien égal au plus petit vecteur propre de la matrice carrée ATA.

Voici le dernier vecteur singulier de notre décomposition SVD:

H_flatten_svd = [-1.65129338e-03 7.48066120e-06 2.92930980e-03 -1.76213405e-05
-1.64173679e-03 9.99991708e-01 -1.60695001e-08 -8.80642885e-09
-1.60653088e-03]

Voici le plus petit vecteur propre de la matrice carrée (A.T)A:

H_flatten_eig = [1.65129338e-03 -7.48072918e-06 -2.92923977e-03 1.76213179e-05
1.64173681e-03 -9.99991708e-01 1.60693694e-08 8.80626428e-09
1.60653111e-03]

Il est possible de constater qu'il sont numériquement égal. Il est à noter que la différence de signe ne sera plus lorsque les vecteurs seront normalisés par leur dernière valeur de -1.61364770e-03 et 1.61364770e-03 respectivement.

Une fois les vecteurs normalisés et mis sous forme de matrice nous obtenons:

H = [[1.02786283e+00 -4.65640672e-03 -1.82337596e+00]
[1.09685663e-02 1.02191424e+00 -6.22454083e+02]
[1.00026089e-05 5.48164305e-06 1.00000000e+00]]

Question 2

Comparez le temps d'exécution de la méthode de décomposition SVD et celle de décomposition en vecteurs propres. Qu'en concluez-vous ?

Le *Tableau 2* présente le temps d'exécution avec la méthode de décomposition en vecteurs propres et la méthode de décomposition SVD pour un nombre K de points correspondants aux deux images. Il est important de noter que la librairie numpy a été utilisée pour effectuer ces calculs (numpy.linalg.eig et numpy.linalg.svd).

Tableau 2: Comparaison entre le temps d'exécution pour la décomposition en vecteurs propres par rapport à la décomposition SVD.

	K = 10	K = 20	K = 200
Décomposition en vecteurs propres	0.00030970 sec.	0.00036860 sec.	0.00032431 sec.
Décomposition SVD	0.00015940 sec.	0.00014421 sec.	0.00416481 sec.

À partir du *Tableau 2*, il est possible de conclure que pour un nombre petit de points correspondants petit la méthode par décomposition SVD est plus rapide. Puisque nous sélectionnons qu'un petite quantité de points correspondants, nous avons choisis K=20, nous avons donc opté pour la méthode par décomposition SVD pour le calcul de la matrice H.

Question 3

Reconstruisez le panorama.

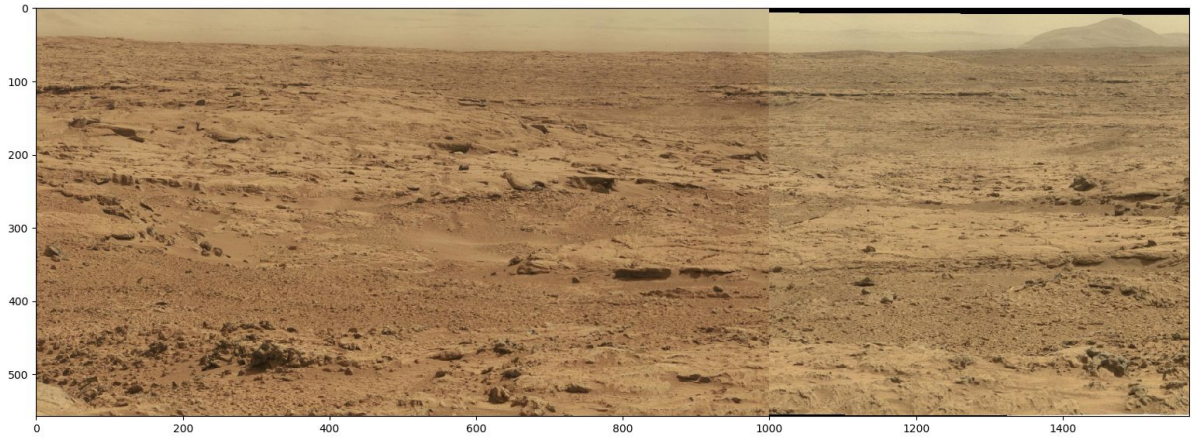

Figure 8: Reconstruction du panorama à partir de l'image gauche et de l'image droite.

Question 4 Bonus

Proposez une méthode pour corriger (automatiquement !) la différence d'exposition entre les deux images. L'implémentation n'est pas demandée, vous avez donc le droit d'utiliser des fonctions pré-existantes si vous souhaitez vérifier votre hypothèse

Une technique possible afin de pouvoir corriger automatiquement la différence d'exposition entre les deux images consiste à faire de la correspondance d'histogramme (*Histogram matching*). La correspondance d'histogramme consiste à faire correspondre l'histogramme d'une image à celui d'une autre image. Dans notre scénario nous avons des images en couleurs nous devons donc faire correspondre les histogrammes de chacune des trois couleurs les constituants: rouge, vert et bleu. Ceci peut être facilement implémenté avec MatLab à l'aide de la fonction *imhistmatch*. En Python, nous avons trouvé une implémentation de cette fonction dans l'article *Histogram matching of two images in Python 2.x?*. Nous avons adaptée cette dernière pour des images en couleurs et voici le résultat final:

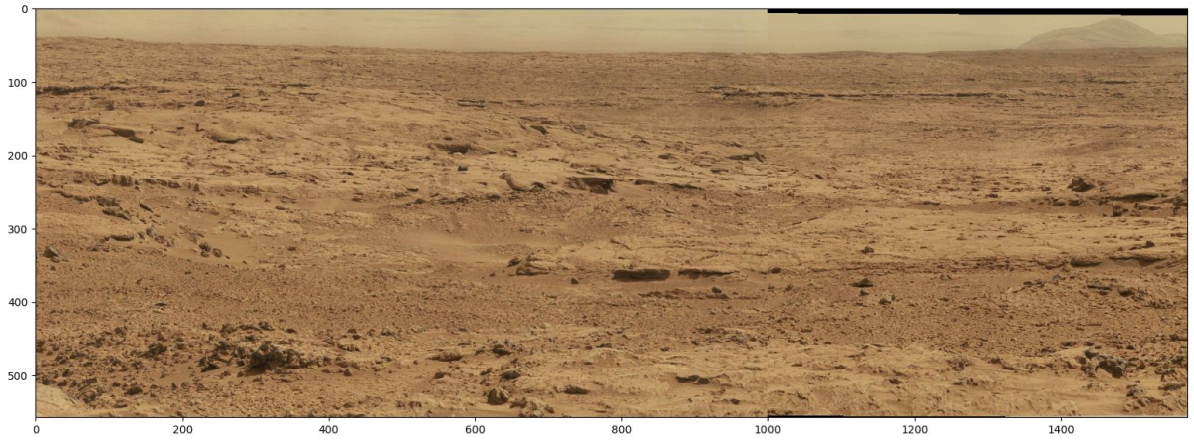

Figure 9: Reconstruction du panorama à partir de l'image gauche et de l'image droite avec correction automatique de la différence d'exposition.