Introduction à la vision par ordinateur
TP3 : Extraction de caractéristiques et mise en correspondance

Thibault Kaspi - thibault.kaspi@student.ecp.fr

20 mars 2016

Résumé

Présentation d'implémentations de différentes méthodes de mise en correspondance des images à l'aide de leurs points d'intérêts respectifs. Le code a été réalisé avec l'aide de deux autres étudiants : Romain Catajar et Léo Hoummady.

1 Mise en correspondance d'images

1. *Sur deux images d'une même scène, calculer les points d'intérêts. Documentez vous sur ces différentes fonctions et utilisez l'une d'entre elles pour calculer des points d'intérêts sur les différentes images à mettre en correspondance.*

   Afin d'avoir des éléments de comparaison pour les questions suivantes, j'ai implémenté plusieurs fonctions de calcul de points d'intérêts. Voici celles que j'ai implémentées :

   — Méthode `SIFT`
   — `goodFeaturesToTrack`
   — `cornerHarris`

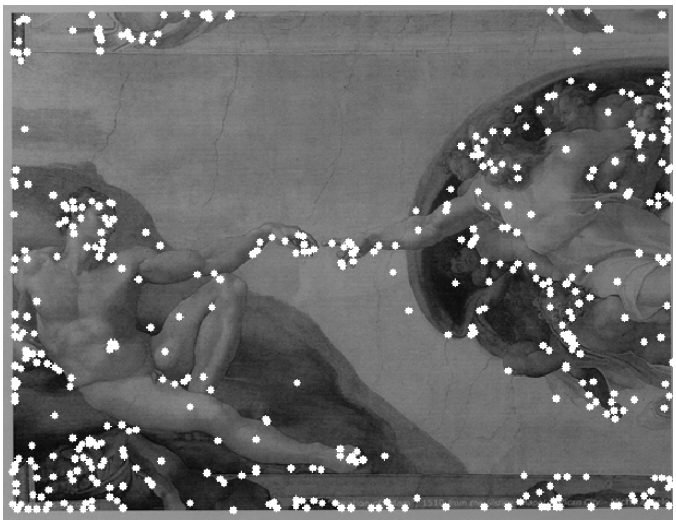

FIGURE 1 – Méthode SIFT sur *La création d'Adam*, l'œuvre de Michel-Ange

2. *Calcul de la similarité entre les deux images. Le principe est de considérer pour un pixel p1 de l'image 1, une fenêtre rectangulaire centrée en p1 et de calculer sa corrélation (sa distance) avec une fenêtre dans la deuxième image. La fonction de corrélation est alors maximum (distance minimum) en p2 correspondant à p1 dans la deuxième image.*

   Le travail consistera donc à :

   (a) *Calculez pour chaque point d'intérêt de l'image gauche sa similarité, avec une des fonctions ci-dessus, avec chacun des points de l'image droite. On affichera le meilleur point correspondant et son score. Un ensemble de pré-traitements sera très certainement nécessaire pour éviter de tester tous les points entre eux.*

   Pour calculer la similarité entre deux images, j'ai procédé de la façon suivante :

   i. Après avoir choisi une fonction de calcul de points d'intérêts (en l'occurrence la méthode `SIFT`), j'applique une fonction de floutage (`cv2.blur`) sur chacune des images afin de lisser les différences.

   ii. Pour chaque point d'intérêt de chaque image, je fais une fenêtre de taille prédéterminée autour du point grâce à la méthode `window_around`.

iii. Pour chaque fenêtre de chaque point d'intérêt, je calcule le coût (fonction de similarité). Si ce coût est inférieur à tous les coûts des fenêtres précédentes, le point d'intérêt associé est mémorisé. Le dernier point d'intérêt à être mémorisé est le meilleur (meilleure similarité entre les deux images). La fonction de coût est calculée grâce à la méthode `cost`.

iv. Enfin, la méthode `plot_windows` est appelée pour afficher la meilleure fenêtre. Dans le cas de la figure 2, `plot_windows` est appelée lorsque le coût est en dessous d'un certain seuil, ce qui permet d'afficher plusieurs points d'intérêt.

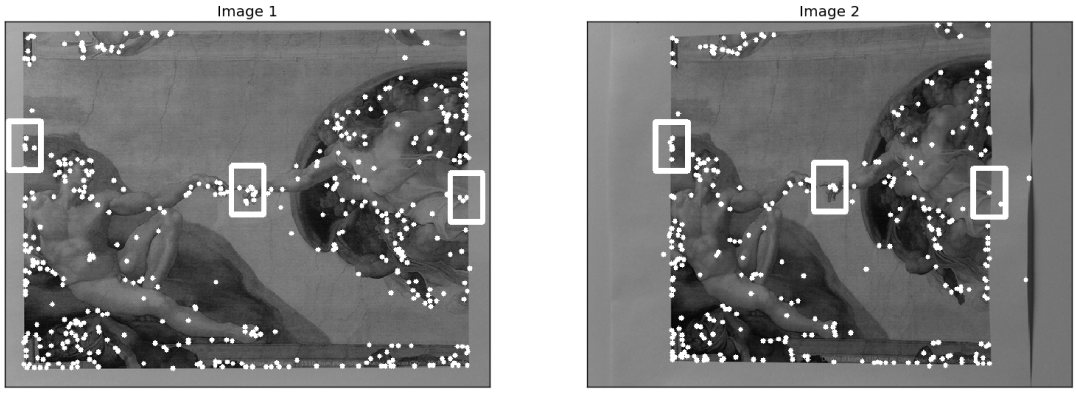

FIGURE 2 – Similarités entre *La création d'Adam* originale et une version avec rotation de 45°. Paramètres : *SAD*, *SIFT*.

Pour la figure 3, j'ai choisi d'utiliser une fonction *SSD*. Le maximum de similarité est atteint pour le point entouré par l'unique fenêtre. Pour ce point, le coût atteint une valeur de 102761.

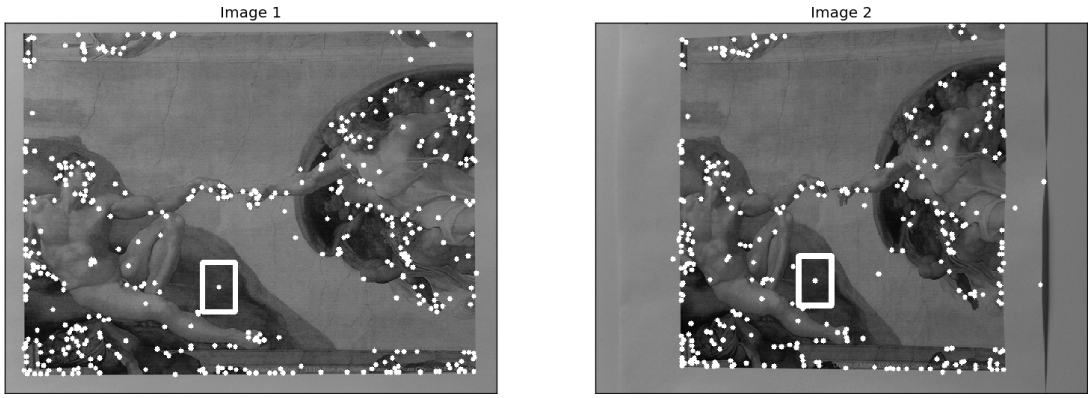

FIGURE 3 – Similarités entre *La création d'Adam* originale et une version avec rotation de 45°. Paramètres : *SSD*, *SIFT*.

(b) *Faire de même mais en échangeant les images de droite et de gauche. Obtient-on le même résultat ? Commentez ces résultats.*

Après avoir échangé les images de gauche et de droite, on observe en figure 4 que le point d'intérêt de similarité maximum est identique au précédent. De même, le calcul du coût donne une valeur strictement identique de 102761.

Pour le coût, le fait que l'on retrouve le même résultat s'explique assez simplement par la symétrie de la fonction de calcul par rapport aux images. Les calculs de points d'intérêt se faisant indépendamment pour les deux images, il est logique que les points résultants soient identiques que l'on prenne les images de gauche à droite ou inversement.

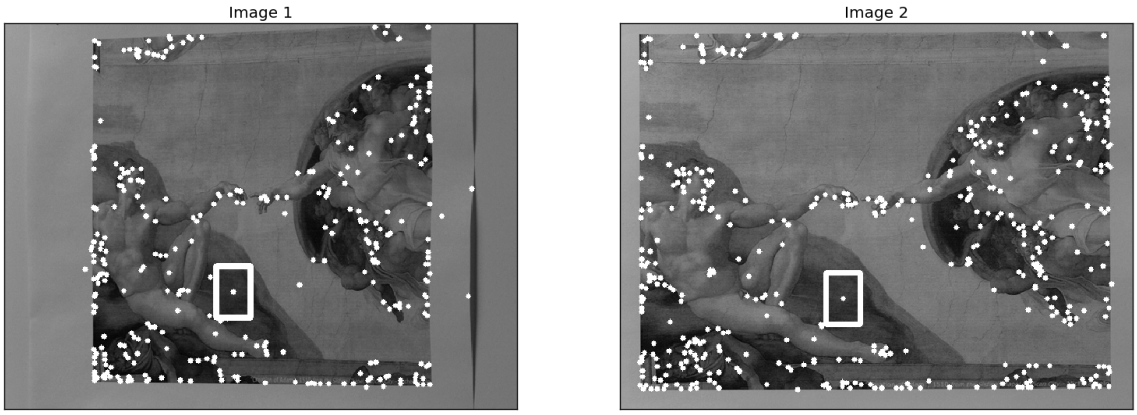

FIGURE 4 – Similarités entre *La création d'Adam* originale et une version avec rotation de 45°. Paramètres : *SSD*, *SIFT*, images inversées.

3. Suite à la mise à jour d'OpenCV en version 3.0 puis 3.1 (version utilisée pour ce TP), les méthodes `BFMatcher` et `FlannBasedMatcher` ne fonctionnent pas correctement. Les environnements testés sont Ubuntu 14.04 et Max OSX El Capitan. Le seul résultat que j'ai réussi à obtenir est pour la méthode `BFMatcher` (figure 5).

   Cette méthode donne de bons résultats pour ce type d'images (ce qui n'est pas le cas pour d'autres catégories d'images que j'ai pu tester). Néanmoins, son mode d'action est assez consommateur en ressources : pour chaque descripteur de la première image, la méthode trouve le descripteur le plus proche dans la deuxième image en les essayant tous un par un.

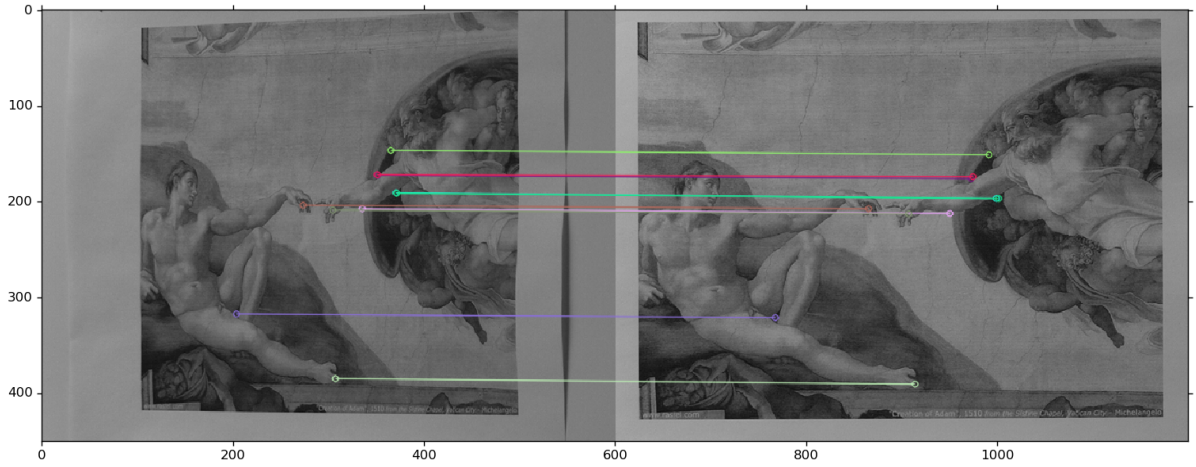

FIGURE 5 – Application de la méthode *BFMatcher* entre *La création d'Adam* originale et une version avec rotation de 45°.

2 Implémenter votre propre détecteur de Harris

2.1 Structure du détecteur

Toutes les méthodes décrites sont situées dans la classe `HomemadeHarris`.

2.1.1 Lissage de l'image par l'opérateur de Sobel

Appelé par la méthode `sobel_transformation`, le lissage de l'image est effectué grâce à la méthode `cv2.Sobel`, afin de garder les gradients selon x et y. Un seuil est nécessaire : par défaut, je l'ai fixé à 5.

2.1.2 Calcul de I_x^2, I_y^2 et I_{xy}

En appelant la méthode `gaussian_blur`, on calcule puis on lisse I_x^2, I_y^2 et I_{xy} avec un filtre gaussien. Un paramètre de lissage, *smoothing_factor*, est nécessaire. Par défaut, celui-ci est de (1, 1).

2.1.3 Calcul de la fonction de Harris

Le calcul de la fonction de Harris se fait en appelant la méthode `harris_function`. La méthode renvoie le résultat du calcul. En figure 6, nous pouvons observer la pertinence de ce détecteur pour détecter les contours dans une image. En revanche, les temps de calculs peuvent être longs.

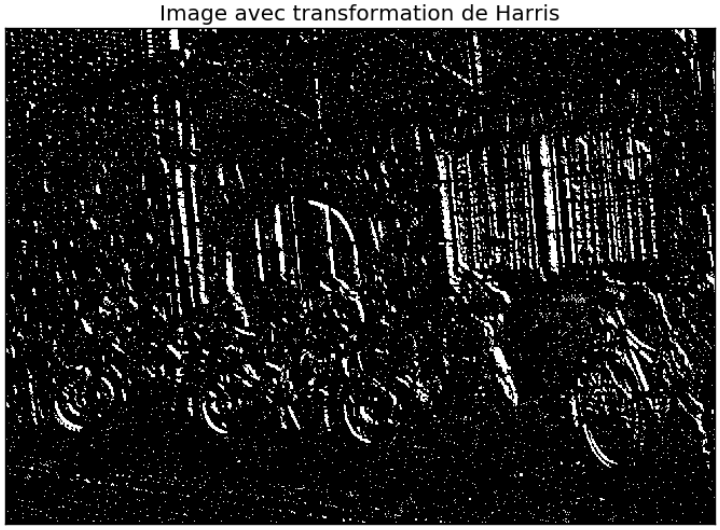

FIGURE 6 – Calcul de la fonction de Harris sur l'image *bike*.
Paramètres : *window_size*=10, *seuil_sobel*=5, *smoothing_factor*=(1, 1)

2.1.4 Extraction des maxima locaux positifs

Pour extraire les maxima locaux positifs, on définit un voisinage (de 3 par défaut) sur lequel on ne va retenir que le maximum. On trie ensuite parmi ces maxima locaux pour en déduire les n maxima globaux de l'image, n étant à choisir. Les méthodes `image_improvement` et `plot` sont appelées ici.

Une fois ces étapes effectuées, on obtient la figure 7. Malgré un bruit de fond assez présent (assez facilement supprimable par ailleurs), la qualité du détecteur semble correcte. Comme pistes d'améliorations, on peut citer l'utilisation de filtres médians contre le bruit par exemple.

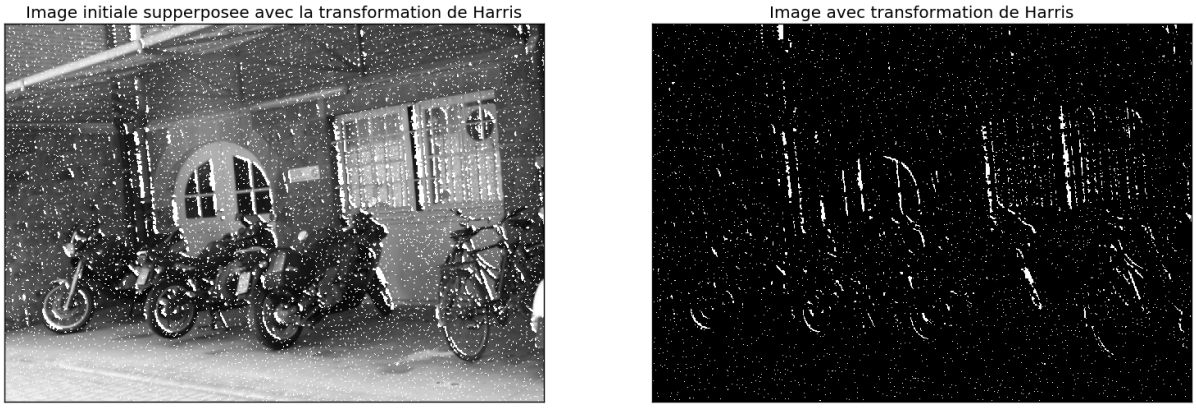

FIGURE 7 – Calcul de la fonction de Harris sur l'image *bike*.
Paramètres : *window_size*=10, *seuil_sobel*=5, *smoothing_factor*=(3, 3), *nb_best_points*=100000