# Cours Introduction à la vision par ordinateur TP 3 : Extraction de caractéristiques et mise en correspondance

ISIA - 2015-2016

L'objectif du travail est de mettre en correspondance des images à l'aide de leurs points d'intérêts respectifs. Vous pourrez utiliser les images de test disponibles ici :

- `http://lear.inrialpes.fr/people/mikolajczyk/Database/det_eval.html`
- `http://www.cmap.polytechnique.fr/~yu/research/ASIFT/dataset_MorelYu09.zip`

## 1 Mise en correspondance d'images

Le principe est le suivant :

1. Sur deux images d'une même scène, calculer les points d'intérêts. Plusieurs fonctions sont disponibles dans OpenCV :
   - `cornerHarris`[1]
   - `cornerSubPix`[2]
   - `goodFeaturesToTrack` qui correspond à l'approche de Shi-Tomasi[3]

[1] `https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners`

[2] `https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners`

[3] `https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html#shi-tomasi`

- `SIFT`[4]

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.

2. Calcul de la similarité entre les deux images.
   Le principe est de considérer pour un pixel $p_1$ de l'image 1, une fenêtre rectangulaire centrée en $p_1$ et de calculer sa corrélation (sa distance) avec une fenêtre dans la deuxième image. La fonction de corrélation est alors maximum (distance minimum) en $p_2$ correspondant à $p_1$ dans la deuxième image.

   Différentes fonctions de dissimilarité sont possibles pour une fenêtre de taille $2N+1 \times 2P+1$

   - SAD : Sum of absolutes differences

   $$SAD(p_1(u_1,v_1),p_2(u_2,v_2)) = \sum_{i=-N}^{i=N}\sum_{j=-P}^{j=P}|I_1(u_1+i,v_1+j)-I_2(u_2+i,v_2+j)|$$

   - SSD : sum of square differences

   $$SSD(p_1(u_1,v_1),p_2(u_2,v_2)) = \sum_{i=-N}^{i=N}\sum_{j=-P}^{j=P}(I_1(u_1+i,v_1+j)-I_2(u_2+i,v_2+j))^2$$

   - ZSSD : Zero mean sum of squared differences

   $$ZSSD(p_1(u_1,v_1),p_2(u_2,v_2)) = \sum_{i=-N}^{i=N}\sum_{j=-P}^{j=P}\left[(I_1(u_1+i,v_1+j)-\overline{I_1(u_1,v_1)})-(I_2(u_2+i,v_2+j)-\overline{I_1(u_2,v_2)})\right]^2$$
   $$\text{avec } \overline{I_1(u_1,v_1)} = \frac{1}{(2N+1)(2P+1)}\sum_{i=-N}^{i=N}\sum_{j=-P}^{j=P}I_1(I_1(u_1+i,v_1+j))$$

[4] `https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro`

- ZNCC : Zero mean normalized cross-correlation

$$ZSCC(p_1(u_1, v_1), p_2(u_2, v_2)) = \frac{1}{\sigma_1.\sigma_2} \sum_{i=-N}^{i=N} \sum_{j=-P}^{j=P} (I_1(u_1+i, v_1+j) - \overline{I_1(u_1, v_1)}).(I_2(u_2+i, v_2+j) - \overline{I_1(u_2, v_2)})$$

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.

(b) Faire de même mais en échangeant les images de droite et de gauche.

(c) Obtient-on le même résultat ? Commentez ces résultats.

3. Mise en correspondance avec OpenCV.
La bibliothèque OpenCV est équipée de plusieurs approches pour la mis en correspondance[5] entre caractéristiques, notamment `cv::DescriptorMatcher`. Documentez-vous sur cette méthode, notamment les différentes variantes `BFMatcher` et `FlannBasedMatcher`, et appliquez la à votre problème de mise en correspondance. Commentez les résultats obtenus.

```
std::string methode=(...);
cv::Ptr<cv::DescriptorMatcher> descriptorMatcher =
        cv::DescriptorMatcher::create(methode);
std::vector<cv::DMatch> filteredMatches12;

// simple match
descriptorMatcher->match( descriptors1, descriptors2, filteredMatches12
```

La fonction `cv::drawKeypoints` permet de dessiner directement des mises en correspondance.

[5] `https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html#matcher`

### 1.1 [optionnel] Mise en correspondance plus avancée

Pour aller plus loin, vous mettrez en oeuvre **une ou plusieurs** des techniques suivantes:

- faire une mise en correspondance croisée, vérifiant pour chaque point d'intérêt que la meilleure correspondance entre l'image 1 et l'image 2 est aussi celle entre l'image 2 et l'image 1. Le principe du code est donné sur le site de Opencv[6]. C'est aussi l'option `crossCheck` du `BFMatcher`.
- utiliser le critère de Lowe (avec un seuil à 0.7) pour la mise en correspondance. On calcule les distance entre un point et plusieurs points correspondants. On ne conserve une correspondance que lorsque le rapport entre la plus petite distance et la seconde est inféreieur à 0.7.
- ajouter une vérification géométrique des mises en correspondance. La fonction `cv::findHomography` permet de trouver une homographie (avec RANSAC notamment).

Au delà de la mise en oeuvre, veillez à **commenter vos résultats**.

## 2 Implémenter votre propre détecteur de Harris

- Calcul de $I_x$ et $I_y$ gradient en $x$ et en $y$ d'une image lissée à l'aide de l'opérateur de Sobel.
- Calcul de $I_x^2$, $I_x^2$ et $I_{xy} = I_x \times I_y$. Lisser avec un filtre gaussien.
- En chaque pixel, calculer la fonction de Harris :$H = detC - \alpha(traceC)^2$ avec $\alpha = 0.04$
- Afficher l'image $H$.

[6] `http://answers.opencv.org/question/15/how-to-get-good-matches-from-\ \the-orb-feature-detection-algorithm/`

- Extraire les maxima locaux positifs sur un voisinage 3 (mettre à 0 tous les points négatifs ou dont la valeur n'est pas supérieure à celle des 8 voisins )
- Extraire les $n$ meilleurs points de Harris (tri par insertion dans un tableau de taille $n$).
- Afficher ces $n$ meilleurs points en dessinant une croix blanche en chaque point sur l'image originale.