Rapport TP2 du Module Vision par Ordinateur
Kasereka Selain

Ce rapport présente notre travail sur le détecteur de points d'intérêt SIFT et le descripteur SIFT. Dans ce travail nous avons utilisé une base d'images nommée **coil-100**. Dans le cadre de ce TP nous avons utilisée la démo disponible sur la page de David Lowe.
Ce travail nous a permis de bien comprendre les notions vues au cours en la matière. Dans les lignes qui suivent nous présentons comment exécuter notre programme que nous avons implémenté en C++ et différents fichiers bash que nous avons écris. Nous allons aussi expliquer comment nous avons fait différents calculs pour le score et la matrice de confusion.

Les démarches générales que nous avions entreprises pour la réalisation de ce TP peuvent être résumées en ces phases ci-dessous :

- création de la base d'apprentissage et de test et aussi les descripteurs respectifs
- maching sur les images de la base d'apprentissage et la base de test ainsi créées
  - calcul du ratio
  - calcul des correspondances entre les images
  - calcul du score
- calcul et création de la matrice de diffusion

1. Fonctionnement de notre programme et autres fichiers bash

Tout d'abord, notons que dans le cadre de ce travail nous avons modifié (ajout des quelques ligne de code) dans le fichier **match.c** de David Lowe. Dans ce dernier nous avons profité pour calculer le score et l'envoyer dans un fichier **fichier_score.txt**, ce fichier contient aussi les correspondances entre les objets.
Nous avons eu besoin de manipuler le répertoire contenant l'ensemble d'images « coil-100 », pour séparer les images da la base d'apprentissage (base_apprentissage) et celle de test (base_test) et aussi pour créer les keypoints pour les deux bases d'images (base_apprentissage_key et base_test_key). Pour cela nous avons utilisé un code bash **creation_key.sh**. Pour exécuter ce code, il est important de le rendre exécutable grâce aux commandes: **chmod +x creation_key.sh** et **chmod 0755 creation_key.sh** en console.

Pour lancer ce code bach il faut utiliser la commande: **sh creation_key.sh**
Notons aussi que cette étape a permis de faire la conversion des images **png** en **pgm** (pour les dossiers d'apprentissage et de test.) et pour cela il a fait appel au code **sift** de David Lowe.

Après il faudra lancer le code de matching, nous avons utilisé un code bash appelant le code **match** de David Lowe (avec quelques modifications que nous avons apportées dans ce code **match.c**). Pour lancer notre bash nous avons utilisé la commande (Veuillez vérifier le droit d'exécution): **sh matchng.sh**. Ce code génère un fichier **fichier_score.txt.**

Le dernier code à lancer est un programme que nous avons écris en C++. Ce code calcul et crée la matrice de confusion. Pour l'exécuter il suffit d'utiliser en console la commande : **./matrice.** Cette exécution appelle le fichier **fichier_score.txt** génère le fichier **matrice_confusion.txt** contenant la matrice de confusion**.**
Soulignons qu'il faut penser à toujours exploiter le **Makefile** en tapant la commande **make** pour générer les fichiers exécutables pour différents codes en C et C++.
Notew bien que vous trouverai dans deux dossiers différents code1 (les différents bash et le match.c de David Lowe modifié) et code2 (le code C++ pour la matrice de confusion).

2. Description des étapes nécessaires pour la résolution du problème

a. création de la base d'apprentissage et de test et aussi les descripteurs respectifs

Comme dit tantôt, avons utilisé une base d'images nommée coil-100. Cette base est grande, il contient exactement 7200 images, ces images sont séparées en 100 classes de 72 images chacune. 50% des images devrait être considérées pour la base d'apprentissage et 50% pour la base de test.
Avec le bash **creation_key.sh** nous avons donc réussi à faire la création des fichiers descripteurs des vecteurs en pré-calculant tous les vecteurs à l'avance, aussi bien pour la base d'apprentissage que celle de validation, ces vecteurs étant stockés dans des dossiers différents comme sus mentionné.
En effet, le code bash a réussi à séparer ces images en deux parties égales comme dit ci-haut, cependant vu les capacités limitées de nos matériels, notamment l'ordinateur avec le disque dur et la mémoire limités, nous avons jugé bon de diminuer le nombre d'images pour le maching et la suite de nos expériences. La taille considérée est donc finalement

une base toujours nommée **coil-100** mais contenant cette fois-ci 800 images de 40 classes avec 20 objets par classe.

En présent nous avons en notre possession les répertoires ci-dessous :

- base_apprentissage : contenant 50% des images en .pgm pour la base d'apprentissage.
- base_apprentissage_key : contenant les descripteurs pour la base d'apprentissage
- base_test : contenant 50% des images en .pgm pour la base de test.
- base_test_key : contenant les descripteurs pour la base de test

b. Maching sur les images de la base d'apprentissage et la base de test ainsi créées

Cette étape est intéressante, elle nous a permis de ressortir la mise en correspondance des points d'intérêts sur différentes images des nos répertoires. Pour calculer les correspondances des descripteurs entre les images, nous avons utilisé la méthode des plus proches voisins, ça revient à calculer le rapport **r** de distances entre le plus proche et le second plus proche voisin, **r** devant être inférieur à un seuil de 1. Dans le cadre de ce travail nous avons considérée un seuil égal à 0.4 car le seuil plus élevé donnait des résultats pqs intéressants : Nous avons utilisé cette formule :

$$ratio = \frac{d_{\text{plus proche}}}{d_{\text{deuxième plus proche}}} < seuil$$

Le matching ainsi effectué nous a donné des résultats des bonnes et mauvaises correspondances que nous présentons avec quelques explications pour chaque cas ci-dessous :

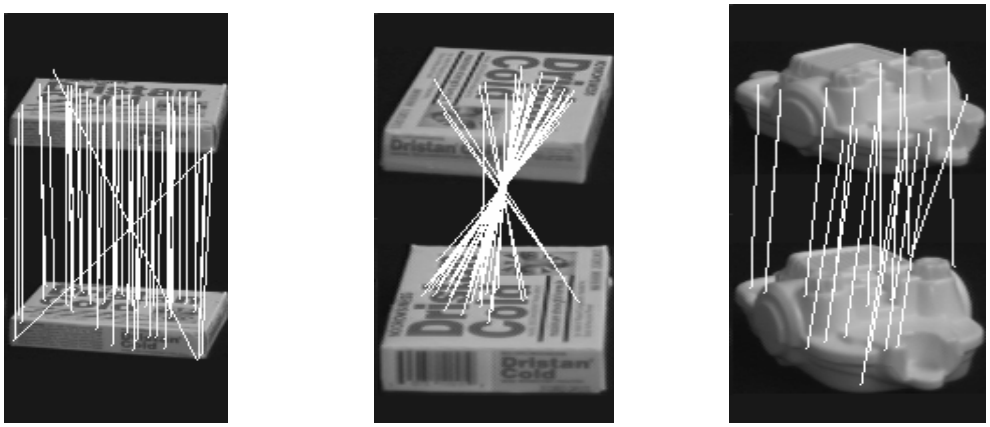
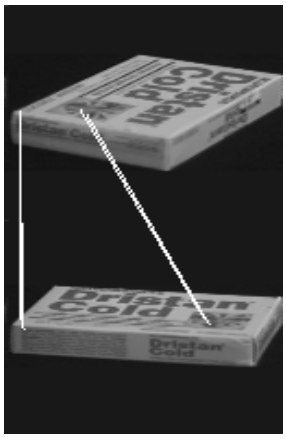


Figure 1. Bonnes correspondances avec les images qui représentent un même objet

Les résultats que nous présentons ci-dessus sont des bonnes correspondances. La méthode a été robuste. En effet, les images représentent un même objet et donc malgré les transformations sur les images (rotations, changement d'échelles, ..), on peu donc voir les correspondances très bien.

Cette image à gauche est représentée le résultat du matching entre deux images représentant deux objets différentes, la méthode est aussi robuste car elle ne trouve aucune correspondance. C'est donc intéressant comme résultat.

Figure 2. Bonnes correspondances (pas de correspondance) avec les images qui représentent des objets différents



Sur cette image à gauche c'est résultat du matching entre deux images représentant deux mêmes objets. La méthode n'a pas pu ressortir beaucoup de correspondances entre les objets pourtant ils sont les mêmes. Nous pensons que cela peut être du au fait que nous avons diminué les nombres initiale d'images de notre base. En effet si la rotation ou l'échelle est forte cela peut être difficile ressortir les correspondances. Ce résultat n'est pas du tout bon.

Figure 3. Correspondance faible avec les images qui représentent le même objet

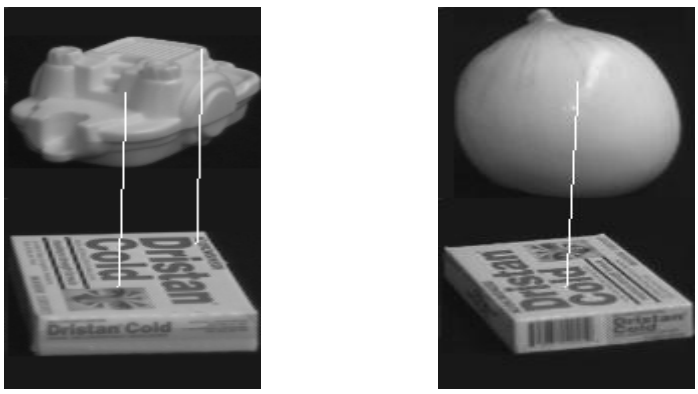


Figure 4. Fausses correspondances entre les images qui représentent des objets différents

Sur ces image ces résultat du matching entre deux images représentant deux mêmes objets. La méthode n'a pas pu ressortir des correspondances entre les objets pourtant ils

sont les mêmes. Nous pensons que cela peut être du au fait que nous avons diminué les nombres initiale d'images de notre base. En effet si la rotation ou l'échelle est forte cela peut être difficile ressortir les correspondances. Ce résultat n'est pas du tout bon.

Figure 5. Aucune correspondance trouvée entre les images qui représentent un même objet.

Sur cette image nous présentons le résultat du matching entre deux images représentant deux mêmes objets. La méthode n'a ressorti aucune correspondances entre les objets pourtant ils sont les mêmes. Cela peut être du a plusieurs facteurs entre autre l'échelle de ces deux images qui est tout a fait différente et aussi une rotation forte. Ce résultat est pas bon tout bon. La méthode a été faible a ce point.

c. Calcul du score

Pour le score nous avons compté le score de correspondances pour l'image inconnue. Nous avons fait cela pour toutes les « images modèles », celles qui se trouvent dans la base d'apprentissage. Nous avons alors considéré que les images qui ont le score le plus élevé déterminent la classe de l'objet inconu ou de la base de test.

Pour calculer le score nous avons utilisee la formule :

$$score = \frac{\#\text{ correspondances réussies}}{\#\text{ descripteurs de l'image modèle}}$$

Pour voir le résultat du score nous l'avons redirigé dans un fichier texte que nous avons nommé fichier_score.txt. Ce fichier contient en effet beaucoup de lignes, chaque ligne reprend la correspondance entre les images inconnues et les images modèles mais aussi le score correspondant. Ci-dessous nous présentons un extrait de ce fichier :

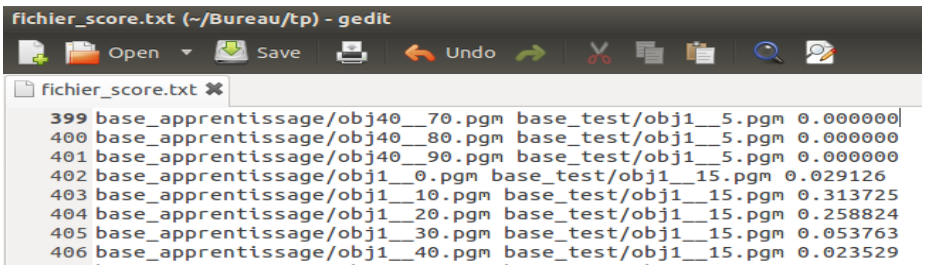


Figure 6. Extrait du fichier_score.txt

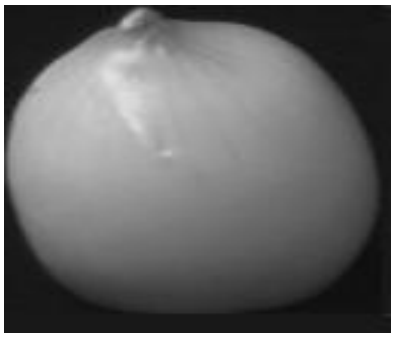
d. calcul et creation de la matrice de confusion

Cette étape peut être vue comme la phase de validation. En effet, étant donné que nous avons considéré un grand nombre d'images, il est difficile de les parcourir toutes à l' œil pour ressortir facilement les correspondances qui fonctionnent bien et d'autres qui fonctionnent moins bien. Pour cela il est nécessaire de faire une matrice de confusion pour voir exactement si tel objet appartient a tel classe, mais aussi ceux qui sont mal classé

Pour écrire cette matrice nous avons écrit un programme en C++ matrice.cpp (voir comment l'exécuter un peu plus haut)

Nous avons constaté qu'il y a des classes qui fonctionnent bien et d'autres qui fonctionnent moins bien. Par exemple :

La classe qui contient ces objets fonctionnes bien dans la pus part de cas. Mais il ya aussi la classe obj2 qui fonctionne relativement bien. Cela est du a la forme des objets ces classes. Elles ont des coins remarquables et cela peut être à la base de cette reconnaissance facile



La classe qui contient ces objets ne fonctionne pas bien. Cela est du a la forme des objets ces classes, en effet la forme de cet objet est rond est a moins de point d'intérêts à ressortir. Noua avons aussi constaté ce même problème pour les objets de la même forme, comme l'image de la tomate par exemple. Il n'est donc pas facile pour la méthode de ressortir les correspondances

Comme nous ne pouvons pas présenter toutes les images, nous vous prions de voir cela dans les exemples présenter dans le point b) sur le matching.

Pour représenter la matrice de confusion, nous avons placé sur les colonnes les classes estimées et sur les lignes les classes réelles. Il nous a fallu alors chaque fois incrémenter

le nombre des objets de la classe une fois l'objet classé.

3. Conclusion

Dans ce TP nous avons travaillé sur le descripteur SIFT. Les résultats obtenus sont intéressants, mais nous devons souligner que nous avons constaté au cours de nos expériences que les la forme de l'image aussi pour une bonne correspondance. La méthode est donc robuste dans des cas spécifiques mais pas dans d'autres cas.

Nous devons en outre souligner que ce TP a des parties plutôt long à réaliser et qui nous ont pris beaucoup de temps, notamment l'étape où il fallait faire le matching entre plusieurs images.
Notons aussi que la démo de David Lowe nous a beaucoup aide et nous a épargné de faire beaucoup de choses. Constatons aussi qu'une transformation trop forte (rotation, point de vue ou échelle) peut altérer la reconnaissance voir renvoyer un résultat nul, mais soulignons que plus on a plusieurs images dans la base d'apprentissage la détection des correspondances entre les images reste élevée.

4. Références

[1] David G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints", 2004.
[2] David G. Lowe, Object Recognition from Local Scale-Invariant Features.
[3] http://www.developpez.net/forums/d1175008/autres-langages/algorithmes/traitement-d-images/comprendre-descripteur-sift/
[4] http://fr.wikipedia.org/wiki/Scale-invariant_feature_transform