SIFT : Description du code

Table des matières

Article étudié Lowe, David. (2004). Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. 60. 91-. 10.1023/B :VISI.0000029664.99615.94
Accessible à : https ://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

Noms et prénoms des membres du groupe de projet Pierre MILLET, Catheline PARISOT, Lucas WANNENMACHER

Résumé du principe, des objectifs de l'algorithme étudié L'objectif de la partie implémentation de l'algorithme était de réussir à détecter les extremas potentiels sur les différences de gaussiennes. Pour cela, nous avons construit l'espace des échelles avec 4 octaves et 5 gaussiennes par octave. Ensuite, nous avons recherché les extremas potentiels sur chaque différence de gaussienne puis nous les avons mis en valeur pour bien les voir à l'affichage.

La partie application met en valeur ce que l'on peut faire avec l'algorithme SIFT notamment grâce aux fonctions d'opencv de python. 5 expériences différentes ont été effectuées lors de la partie application. D'abord, la mise en évidence des enjeux du SIFT avec l'image Lena, notamment par rapport à l'invariance par rotation et au changement d'échelle. Ensuite, 3 différents scénarios de reconnaissance d'objets ont été effectués. Enfin, nous avons implémenté une reconnaissance d'objet en direct avec une acquisition vidéo qui met en valeur le potentiel de l'algorithme SIFT.

Structure de l'archive zip déposée :
Fichier implémentation : Implementation_SIFT_MILLET_PARISOT_WANNENMACHER.ipynb
Fichier application : Application_SIFT_MILLET_PARISOT_WANNENMACHER.ipynb
Fichier description : DescriptionCode_SIFT_MILLET_PARISOT_WANNENMACHER.pdf

Première partie

Implémentation de l'algorithme

1 Description

Le code se découpe en deux grandes sous-parties. Tout d'abord, il y a la construction de l'espace des échelles puis la détection des extremas dans les différences de gaussienne générées.

1.1 Etape 1 : Création de l'espace des échelles

On génère tout d'abord l'image qui est ici Lena après avoir importé nos bibliothèques utiles.

La première fonction `generationOfGaussianAndDoGImages(img, sigma, k)` permet de créer l'espace des échelles en appliquant des filtres gaussiens. Elle permet aussi de calculer les DoG (différences de gaussiennes) qui nous seront utiles pour la deuxième étape implémentée. Pour l'application du filtre gaussien, on utilise `cv2.GaussianBlur` qui prend en paramètre l'image, la taille du masque gaussien (5,5) et le facteur d'échelle σ qui correspond à l'écart-type de notre distribution gaussienne. Cette fonction permet de générer un octave, étage sur lequel l'image a la même dimension. Les valeurs de k et σ sont arbitraires. Nous avons pris des valeurs entre 1 et 2 pour rester raisonnable, nous souhaitions voir une évolution mais pas trop grande afin que les images ne soient pas trop floutées.

Algorithme 1 Exemple pour la première DoG

```
Gaussian1 = cv2.GaussianBlur(img, (5,5), sigma)
Gaussian2 = cv2.GaussianBlur(img, (5,5), sigma*k)
DOG1 = Gaussian2 - Gaussian1
```

Après avoir créé cette fonction, on génère 4 octaves. Pour chaque octave, on redimensionne l'image par 2 grâce au resize. 4 octaves semble être un bon compromis car on va déjà voir que pour la 4ème octave, l'image est en $64 * 64$ pixels et on ne trouve déjà presque plus d'extremas. 5 images gaussiennes semble aussi être raisonnable car nous aurons, de cette façon, 4 différences de gaussienne par octave ce qui semble être un chiffre minimal et optimal pour notre application.

Algorithme 2 Exemple pour la première octave

```
DOG1_1, DOG2_1, DOG3_1, DOG4_1, Gaussian1_1, Gaussian2_1,\
Gaussian3_1, Gaussian4_1, Gaussian5_1 = generationOfGaussianAndDoGImages(img, sigma, k)
```

1.2 Etape 2 : détection des extremas dans les différences de gaussiennes

A partir des DoG créées par la fonction `generationOfGaussianAndDOGImages`, on crée la fonction `findExtremas(DoG_prev,` qui va permettre de trouver les extremas potentiels pour une DoG donnée. Pour cela, on a aussi besoin des différences de gaussiennes précédentes et suivantes sur le même octave. C'est pourquoi nous aurons des résultats seulement pour la deuxième et la troisième DoG de chaque octave. Dans la fonction `findExtremas`, on cherche les coordonnées des extremas potentiels en parcourant toute l'image et en analysant pour chaque pixel, 26 voisins (8 sur la DoG actuelle et 9 sur la précédente et la suivante).

On sélectionne un pixel s'il est supérieur au max de neighbours (voisins) ou inférieur au min de neighbours. On obtient finalement les coordonnées des extremas.

Algorithme 3 Sélection des extremas selon la condition ci-dessus

```
if (DoG2[i,j].all() > max_neigh) or (DoG2[i,j].all() < min_neigh):
      extremas_x.append(i)
      extremas_y.append(j)
```

On calcule ensuite les positions des extremas pour nos DoG, cela prend d'ailleurs un peu de temps à compiler sachant le nombre important de calculs et de boucles à réaliser.

Algorithme 4 Exemple de recherche d'extremas pour la première DoG

```
DOG2_1_extremas_x, DOG2_1_extremas_y = findExtremas(DOG1_1, DOG2_1, DOG3_1)
DOG3_1_extremas_x, DOG3_1_extremas_y = findExtremas(DOG2_1, DOG3_1, DOG4_1)
```

Pour bien repérer les extremas potentiels dans les DoG, on propose la fonction

`highlightExtremas(DOG, DOG_extremas_x, DOG_extremas_y)` qui va permettre de créer de petits cercles rouges autour des extremas (grâce à la fonction `cv2.circle`) et ainsi plus simplement les identifier. Finalement, on peut afficher nos différences de gaussienne avec les extremas reperés.

2 Analyse

Les résultats obtenus sont, à notre niveau, satisfaisants. Nous n'avons pas eu grand mal à générer l'espace des échelles après s'être efforcé de bien comprendre le principe. La pyramide de gaussiennes ainsi créée est conforme aux attentes et montre bien l'évolution attendue. La génération des différences de gaussiennes s'est aussi faite sans problème comme nous avions déjà pu l'étudier en TP. La recherche des extremas a, quant à elle, été plus compliquée pour nous mais nous avons tout de même pu obtenir un résultat satisfaisant qui met bien en valeur les points d'intérêt potentiels des DoG.

3 Pistes d'améliorations

Les pistes d'améliorations sont ici nombreuses car, en effet, une petite première partie de l'algorithme SIFT a été implémentée. Il aurait été intéressant dans le cadre d'un projet de plus longue durée, de coder les étapes suivantes comme l'élimination des extremas non-pertinents (d'où l'expression depuis le début d'extremas « potentiels »), la localisation de ceux-ci sur l'image de base et non pas que sur les DoG ou encore le calcul d'orientation des points que nous avons aussi pu expliquer lors de l'oral.

Deuxième partie

Applications de l'algorithme avec les fonctions d'OpenCV

La partie application peut se diviser en 3 parties : la prise en main avec l'image Lena, les 3 scénarios de reconnaissance d'objets dans un décor et la reconnaissance d'objet en direct dans un décor.

4 Description

La prise en main consistait à calculer, afficher puis comparer avec un matching les points d'intérêts et descripteurs de deux images : l'image Lena originale et cette même image sur laquelle on a effectué une rotation et un changement d'échelle. Dans la seconde partie est présentée une reconnaissance d'un jeu vidéo dans un décor.

La démarche a été la suivante : calculer les points d'intérêts et descripteurs d'une image d'entraînement et d'une image requête du jeu dans un décor, puis effectuer un matching entre ces deux images. On présente ensuite la même expérience avec deux autres objets, un Rubik's Cube et une pochette d'album, que l'on avait fait en amont de cela, pour montrer notre démarche pour arriver à un résultat concluant ainsi qu'à un aperçu des limites.

Enfin, la partie détection d'objets en direct consistait, à l'aide d'une acquisition vidéo, à changer en boucle l'image requête et, si l'objet était détecté, à tracer une polyligne autour de l'objet. Le seuil de détection de l'objet est un seuil constant fixé arbitrairement au début et correspondant au nombre de matchs corrects sur l'objet détecté.

5 Analyse

Les résultats obtenus avec la comparaison des images de Lena furent conformes à nos attentes : le matching était presque parfait, ce qui mets en évidence l'invariance par rotation et changement d'échelle du SIFT.

La détection du jeu vidéo « Uncharted » dans un décor fut très concluante : le taux de matches corrects était très élevé, et très nettement mis en évidence avec le matching.

FIGURE 1 – Mise en correspondance des descripteurs

En revanche, les deux expériences présentées ensuite, qui ont été effectuées en amont, ont été moins concluantes. Pour l'objet « pochette d'album », la raison majeure du relatif échec était que l'objet était très sombre et donc que certains détails, présents dans l'image d'entrainement, ne l'étaient pas dans l'image cible. Ces deux expériences mettent en valeur des limites, peut être pas du SIFT, mais de son utilisation de telle manière avec python/OpenCV.

Enfin, la détection d'objet en direct de l'objet Uncharted est impressionnante par son efficacité. En effet, le jeu est détecté avec une grande efficacité, et l'algorithme ne se laisse pas tromper par des objets qui lui ressemblent.

FIGURE 2 – Détection du jeu Uncharted en vidéo

Petit bémol : l'acquisition et le traitement en direct est très gourmand pour le pc, mais cela peut être résolu en ajoutant un délai d'attente dans la boucle, c'est-à-dire en baissant la fréquence des images.

6 Pistes d'amélioration

En piste d'amélioration, nous aurions pu travailler avec les 3 canaux d'origine. En effet, la norme dans l'utilisation du SIFT avec python/OpenCV est de convertir l'image en nuances de gris et ensuite d'effectuer le traitement : de l'information est donc perdue. L'utilisation des 3 canaux aurait pu permettre une meilleure fiabilité, car la quantité d'information disponible aurait été plus importante.