30/09/2013

Guénon Marie et Favreau Jean-Dominique

Master SSTIM

Compression des images numériques

Compte rendu TD1 : transformée en ondelettes, filtre de Haar

Table des matières

[Préambule : 2](#_Toc368917994)

[Librairies : 3](#_Toc368917995)

[Filtre de Haar, Analyse : 4](#_Toc368917996)

[Reconstitution, Synthèse : 6](#_Toc368917997)

[Inversion directe 6](#_Toc368917998)

[Filtrage 7](#_Toc368917999)

[Récursivité 8](#_Toc368918000)

[Annexes : 9](#_Toc368918001)

**But :** réaliser une transformée en ondelettes (analyse / synthèse) au moyen du filtre de Haar.

# Préambule :

Après une première approche avec Scilab, nous avons préféré utiliser le langage de programmation C++ pour implémenter ce projet. En effet, ce langage nous permet d’avoir une approche simplifiée du traitement d’image ainsi que des calculs optimisés : les calculs matriciels et les itérateurs sont déjà implémentés et optimisés pour le genre de calculs que nous allons devoir effectuer. Nous obtiendrons donc ainsi de meilleurs résultats, autant au niveau qualitatif, qu’au niveau de la rapidité d’obtention.

# Librairies :

De manière native, C++ ne sait pas charger d’image. Nous avons donc du installer la librairie :

**OpenCv2**



Cette librairie permet de charger, afficher et traiter les images.



# Filtre de Haar, Analyse :

## Méthode directe :

Une fois l’image à traiter chargée, nous allons la traiter. Pour ce faire, nous allons la parcourir par bloc de quatre pixels, que l’on nommera. On applique alors les quatre filtres de Haar à chaque point considéré de notre image de départ :

* Moyenne des pixels avoisinants, on applique le filtre : et on obtient l’image résultante :





* Gradient X, on applique le filtre : et on obtient l’image résultante :





* Gradient Y, on applique le filtre : et on obtient l’image résultante :





* « Gradient diagonal », on applique le filtre : et on obtient l’image résultante :





Ce qui donne, une fois touts les filtres réunis ensembles, on obtient[[1]](#footnote-1) :



## Filtrage

Avant de filtrer l’image, nous devons lui appliquer un traitement préalable. En effet, pour éviter les effets de bords et les problèmes aux limites, nous devons insérer une ligne et une colonne supplémentaire à la fin de l’image. Pur cela, nous insérons par miroir (copie du pixel précédent) ces lignes à droite et en bas de l’image.

Après quoi nous appliquons les quatre filtres vus précédemment mais sur l’intégralité de l’image :

* Moyenne des pixels avoisinants:



* Gradient X   :



* Gradient Y :



* « Gradient diagonal »  :



Pour finir, nous effectuons le sous échantillonnage sur les quatre filtres obtenus : nous prenons un pixel sur deux en fonction du filtrage effectué :

* Moyenne des pixels avoisinants:  
  On prend les colonnes et les lignes impaires.
* Gradient X  :  
  On prend les colonnes paires et les lignes impaires.
* Gradient Y :   
  On prend les colonnes impaires et les lignes paires.
* « Gradient diagonal »  :  
  On prend les colonnes et les lignes paires.

# Reconstitution, Synthèse :

## Inversion directe

Comme vu précédemment, nous avons une image de départ et nous l’avons parcours par blocs de quatre pixels que nous avons nommé. Cette image a été analysée en quatre images grâce aux filtres décrits ci-dessus :

Pour reconstituer l’image de départ, Il suffit d’inverser le système présenté ici et nous obtenons les formules suivantes :



En appliquant ces formules sur les filtres obtenus précédemment, on obtient à gauche l’image d’origine et à droite l’image reconstruite :

 

## Filtrage

# Récursivité :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre d’itération | 0 | 1 | 2 | 3 |
| Résultat obtenu |  |  |  |  |

# 

# Annexes :

1. Les images qui sont affichées ici ont été retouchées pour pouvoir être visualisable. En effet, une fois les filtres appliqués, les valeurs obtenues ne sont pas des images et doivent être translatées et étirées entre 0 et 255. [↑](#footnote-ref-1)