INF4710
Introduction aux technologies multimédia

H2016 - Travail pratique #3
Indexation de fichiers multimédia : détection de changement de scène

---

Objectifs :

- Permettre à l'étudiant de se familiariser avec la convolution d'un signal 2D ainsi qu'avec les algorithmes de détection de changement de scène dans une séquence vidéo

Remise du travail :

- Au plus tard, le 12 avril, 14h00 sur Moodle – **aucun retard accepté**

Références :

- Voir les notes de cours sur Moodle (indexation contenu pictural)

Documents à remettre :

- Vos scripts/fonctions/sources ainsi qu'un rapport (**.pdf**) dans une archive (.zip/.7z/...) nommée convenablement

Autres directives :

- Les en-têtes et les commentaires sont fortement suggérés dans le code… (plus façile de vous donner des points quand tout est brisé!)

Présentation

L'objectif de ce travail pratique est d'implémenter une méthode de détection de changements entre scènes dans une vidéo par détection et mise en correspondance d'arêtes. Pour ce faire, il est nécessaire d'extraire l'image de gradient de chacune des trames de la séquence et d'y effectuer un seuillage pour y identifier les arêtes présentes.

Contrairement à ce qui vous était demandé aux deux premiers TPs, pour le TP3, un poids beaucoup plus important de la notre finale est accordé à votre rapport et à vos résultats. Traitez votre rapport comme un compte-rendu d'implémentation avec résultats de tests **que vous enverriez à un client**. Notez toutefois que ce dernier **n'y connaît absolument RIEN en programmation**, alors ne recopiez pas simplement votre code! Vous devrez aussi y discuter vos choix stratégiques, et les avantages/inconvénients de vos méthodes (**voir barème à la fin**).

Tel que décrit à travers les pages 23 à 33 du chapitre sur l'indexation du contenu pictural provenant des notes de cours, vous aurez d'abord à implémenter une fonction de convolution d'image à l'aide d'un filtre de Sobel. Par la suite, vous aurez à développer (**par vous-même**) une fonction de seuillage des images de gradient obtenues pour y détecter les arêtes principales de la scène. Ces arêtes devront ensuite être dilatées à l'aide d'une opération morphologique. Finalement, vous aurez à calculer les ratios d'arêtes partagées par des paires d'images à travers une séquence vidéo (ρ_{in} et ρ_{out}), et déterminer quelle stratégie utiliser pour maximiser vos chances de détecter les changements de scènes dans n'importe quelle vidéo. Le travail demandé pour chacune de ces étapes est décrit dans les sections suivantes.

Convolution d'images

Ici, nous vous demandons d'implémenter une fonction qui calcule **la carte des forces de gradient normalisée** à l'intérieur d'une image couleur. Pour ce faire, vous devez aussi implémenter une deuxième fonction pour calculer la réponse de convolution d'une image à l'aide d'un noyau quelconque fourni en paramètre. Vous devez suivre la stratégie énoncée aux pages 53 à 55 du même chapitre des notes de cours dans le but d'obtenir une carte des forces calculées en X et en Y. En gros, le prototype de la fonction principale est le suivant :

```
function [ gradient ] = fg_norm_sobel( image )                    (Matlab)
```

```
cv::Mat fg_norm_sobel(const cv::Mat& image) { ... }               (C++)
```

Où '*image*' (la matrice en entrée) contient un ou trois canaux, et '*gradient*' (la matrice en sortie) est de taille identique, possède le même nombre de canaux, mais est de type 'float', avec des valeurs **normalisées entre 0.0 et 1.0** (inclusivement).

La fonction '*fg_norm_sobel*' appellera elle-même, deux fois pour chaque canal de '*image*', la fonction de convolution avec le prototype suivant :

```
function [ response ] = convolution( image_1ch, kernel )                    (Matlab)
```

```
cv::Mat convolution(const cv::Mat& image_1ch, const cv::Mat& kernel) { ... }    (C++)
```

...soit une fois pour l'analyse avec un noyau de Sobel en X, et une fois en Y. Notez qu'ici, les matrices '*image_1ch*' données en entrée doivent bien posséder un seul canal. Les noyaux de Sobel à utiliser sont de taille 3x3 et sont décrits ci-dessous :

-1	0	1
-2	0	2
-1	0	1

S_x : Noyau Sobel 3x3

1	2	1
0	0	0
-1	-2	-1

S_y : Noyau Sobel 3x3

Par la suite, la formule à utiliser dans '*fg_norm_sobel*' pour obtenir la norme (i.e. la « force ») du gradient en un point (i,j) dans l'image (et pour un seul canal à la fois) à partir des deux réponses de gradients en X et en Y est la suivante :

$$F_G(i,j)=\sqrt{G_x^2(i,j)+G_y^2(i,j)}$$

...où G_X et G_Y sont les réponses de gradient en X et Y pour le pixel (i,j). Finalement, vous devez faire la normalisation 0-1 (aussi appelée 'min-max') de toutes les valeurs F_G, canal par canal, avant de les retourner comme sortie de '*fg_norm_sobel*'. Allez voir https://en.wikipedia.org/wiki/Feature_scaling pour une description!

Bien entendu, nous nous attendons à ce que vous implémentiez ces deux fonctions au complet, sans utiliser les fonctions de convolution déjà disponibles dans Matlab/OpenCV. Vous pouvez toutefois tester le fonctionnement de vos méthodes en comparant vos résultats à ceux de 'imgradient' (Matlab), ou à ceux de cet exemple (OpenCV). Attention aux types/intervalles lors de vos comparaisons!

Seuillage des images de gradients

Dans cette partie, vous devez vous-même choisir une stratégie pour transformer l'image des forces de gradient (qui devrait contenir un ou trois canaux) en une image binaire (donc un canal, et deux valeurs possibles) contenant une valeur non-nulle uniquement là où une arête « importante » est présente. Il est clair qu'il faut utiliser une opération de seuillage pour y arriver, mais c'est à vous de choisir l'opération ainsi que le (ou les) seuils à utiliser, au besoin. Dans votre rapport, décrivez bien l'approche que vous avez choisie et parlez des avantages et des inconvénients de celle-ci. Ultimement, vous devriez être capable d'obtenir une transformation semblable à celle de l'image ci-dessous.

Image couleur originale

Image binaire des arêtes détectées

Dilatation des arêtes trouvées

Pour dilater les arêtes des images binaires obtenues précédemment, vous devez implémenter une fonction ayant le prototype suivant :

```
function [ dilated_image ] = dilate_image( image, N )                    (Matlab)
```

```
cv::Mat dilate_image(const cv::Mat& image, size_t N) { ... }             (C++)
```

... où l'image binaire à traiter, '*image*', est fournie en même temps que la taille de dilatation (notée *N*, et qui est toujours un entier impair). L'opération de dilatation est relativement simple : vous n'avez qu'à parcourir tous les pixels de l'image, et pour chaque pixel ayant une valeur non nulle, vous devez copier cette valeur à tous les pixels présent dans un voisinage de taille N x N centré sur le pixel original. Cette approche est en fait une simplification de celle expliquée dans le cours (i.e. dilatation avec un cercle ou un losange). Le résultat que vous devriez obtenir, une fois cette opération complétée, devrait ressembler au suivant (pour N=5) :

Image binaire originale

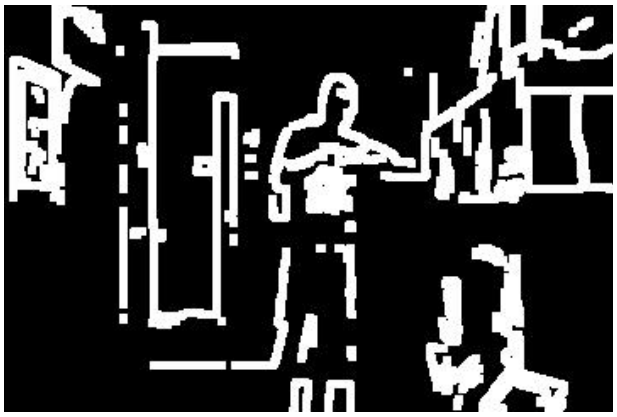

Image binaire dilatée

Calcul du ratio des arêtes partagées

Une fois que vous disposez d'images d'arêtes dilatées et non dilatées, vous pouvez enfin calculer les valeurs de ρ_{in} et ρ_{out} en utilisant les formules suivantes (tirées des notes de cours) :

$$\rho_{in} = 1 - \frac{\sum_{x,y} D(x,y,t)E(x,y,t+1)}{\sum_{x,y} E(x,y,t+1)}$$

$$\rho_{out} = 1 - \frac{\sum_{x,y} E(x,y,t)D(x,y,t+1)}{\sum_{x,y} E(x,y,t)}$$

...où *E* est une image binaire d'arêtes non dilatée, et *D* est une image binaire d'arêtes dilatée. Puisque ces deux formules ne diffèrent que par la nature des images utilisées, elles peuvent toutes deux être calculées à l'aide d'une fonction commune, qui devra être implémentée sous le prototype suivant :

```
function [ p ] = edge_ratio ( A, B )                                    (Matlab)
```

```
float edge_ratio ( const cv::Mat& A, const cv::Mat& B ) { ... }         (C++)
```

...où *A et B* peuvent contenir n'importe quelle combinaison d'images binaire, et où ρ est calculé à partir de la formule suivante :

$$\rho = 1 - \frac{\sum_{x,y} A(x,y)B(x,y)}{\sum_{x,y} B(x,y)}$$

Détection de transitions dans une séquence vidéo

Pour cette partie, vous devez créer une fonction d'analyse capable de traiter la séquence vidéo fournie avec l'énoncé ('TP3_video.avi' disponible sur Moodle) pour y détecter les transitions (**coupures et fondus**). Votre analyse devra faire usage de toutes les fonction implémentées précédemment. La stratégie d'analyse à suivre est libre à vous; vous avez quelques exemples dans les notes de cours de ce qui peut être fait avec ρ_{in} *et* ρ_{out} en fonction du temps, mais vous pouvez décider d'y ajouter d'autres composantes (au besoin). Notez que votre stratégie d'analyse devrait pouvoir être appliquée à n'importe quelle séquence, et pas seulement celle fournie avec l'énoncé! Bref, n'allez pas 'hard-coder' des indices propres à 'TP3_video.avi' dans votre analyse...

Au final, dans votre rapport, vous devez (minimalement) présenter un graphique des valeurs de ρ_{in} *et* ρ_{out} en fonction du temps pour 'TP3_video.avi', et les numéros de trames où des changements ont pu être détectés. Notez qu'il est possible que vous n'arriviez pas à détecter toutes les transitions de 'TP3_video.avi' (deux coupures, deux fondus) **si votre stratégie est trop simple, ou si vos paramètres sont mal choisis.** Dans ce cas, vous devez **obligatoirement** en détailler la cause (selon vous) dans votre rapport, et offrir une nouvelle piste de solution (théorique seulement) qui pourrait régler le problème (voir le barème). Si vous détectez bien toutes les transitions aux bons endroits, vous pouvez ignorer cette partie de la discussion (qui vaut **2.5 pts**).

Notez que pour éviter d'avoir à relancer l'analyse de la séquence vidéo pendant que vous testez/ajustez votre algorithme de détection de transitions, vous pouvez sauvegarder sur le disque toutes les valeurs de ρ_{in} *et* ρ_{out} obtenues lors d'une première analyse, et les recharger par la suite. Cela ne sera probablement pas nécessaire en C++, mais en Matlab, ça pourrait vous sauver pas mal de temps de calcul...

Barème : (total sur 20 pts)

1. **Implémentation/fonctionnement** : (sur 6 pts)
    - fg_norm_sobel, convolution = 2 pts
    - dilate_image = 2 pts
    - edge_ratio = 2 pts
2. **Rapport** : (sur 14 pts – avec page titre et images, **~6 à 8 pages**)
    - Présentation modules de convolutions, seuillage, dilatation = 2.5 pts (~2 pages)
    - Discussion avantages/inconvénients, et rôle des paramètres (si présents) pour stratégie de seuillage = 2 pts (½ à 1 page)
    - Présentation, discussion avantages/inconvénients, et rôle des paramètres (si présents) pour stratégie de détection de transitions = 3 pts (~1 page)
    - Présentation résultats de détections pour 'TP3_video.avi' = 1.5 pts (~½ page)
    - Cause détections mauvaises/manquantes, solution possible = 2.5 pts (~½ page)
    - Propreté, formatage, lisibilité = 2.5 pts

Références supplémentaires

- Aide-mémoire (« Cheat sheet ») Matlab :
    - http://web.mit.edu/18.06/www/Spring09/matlab-cheatsheet.pdf
- Guide complet Matlab :
    - http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf
- OpenCV
    - http://opencv.org/
    - http://docs.opencv.org/doc/tutorials/tutorials.html