École Polytechnique de Montréal - Département de Génie Informatique et Génie Logiciel

INF4710 : Introduction aux technologies multimedia - H2016

TP3 - Indexation de fichiers multimédia : détection de changement de scène

Javid Mougamadou 1809167

Christine Chand 1572129

10 avril 2016

Soumis à :

Pierre-Luc St-Charles

Table des matières

1 Présentation modules de convolutions, seuillage, dilatation

1.1 Convolution d'images

1.1.1 Opération de convolution

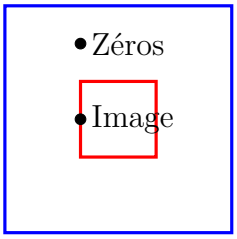
Voici l'en-tête de notre fonction de convolution :
function [response] = convolution(image_1ch, kernel)

Cette fonction prend en entrée une image (I) avec un seul canal et une matrice de noyau de taille 3x3 (K). Voici la formule de convolution de matrice utilisée :

$$C(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(x,y) * K(x-i, y-i)$$

Nous retrouvons une symétrie des indices lors du calcul du produit des valeurs de la matrice. C'est pourquoi, nous allons effectuer une rotation de 180 degrés sur la matrice du noyau dans le but d'obtenir un produit des valeurs de matrices simplifiées .

Si nous avons une matrice correspondant à l'image d'entrée de taille (n x p) alors la matrice obtenue en sortie par l'opération de convolution doit être de la même taille (n x p). Cependant, nous rencontrons un problème lorsqu'il faut calculer les valeurs qui sont sur les bords de notre matrice. C'est pourquoi, il faut ajuster notre matrice de manière à ce que toutes les valeurs soient calculées en ayant une matrice de taille (n+2 x p+2) avec des 0 vers les frontieres.



En notant I_{ajuste} la matrice ajuste avec des 0 aux frontières et H la matrice de noyau ayant obtenue une rotation de 180°, nous avons donc la formule suivante :

$$\forall (x,y) \in (2, n+1)(2, p+1), C(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I_{ajuste}(x,y) * H(x,y)$$

Voici l'image obtenue après convolution du premier canal de l'image avec le noyau S_x (voir plus bas) :

FIGURE 1 – Image Originale

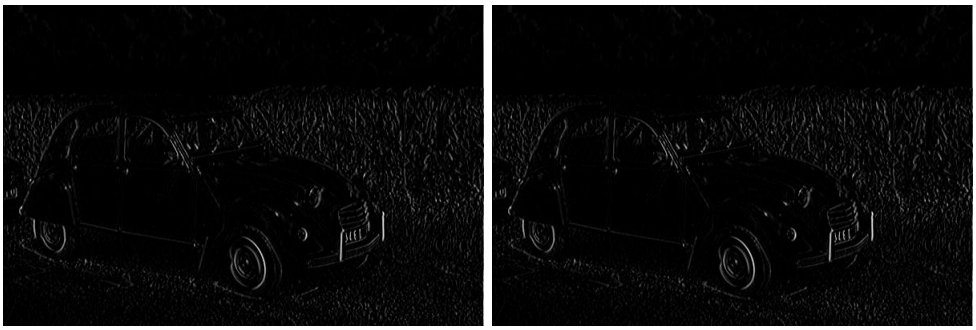


FIGURE 2 – Image résultant d'une convolution avec comparaison avec la fonction *conv2* de Matlab

1.1.2 Opération de calcul des forces de gradient normalisées

Voici l'en-tête de notre fonction de calcul des forces de gradient normalisées :
function [gradient] = fg_norm_sobel(image)

Cette fonction prend en entrée une image (I) avec un seul canal ou trois canaux . Pour chaque canal, nous allons appliquer la convolution grâce à la fonction précedente avec les noyaux de *Sobel* suivants :

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \text{ et } S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Ce qui nous donne les formules suivantes :

$$G_x(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(x,y) * S_x(x-i, y-i)$$

$$G_y(x,y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(x,y) * S_y(x-i, y-i)$$

Ensuite nous calculons les forces de gradient en utilisant la norme :

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

Une fois que les forces de gradient sont calculées pour toutes les valeurs, nous normalisons les valeurs avec la méthode *min-max* pour obtenir des valeurs dans l'intervalle [0,1]. Voici l'image obtenue après calcul des forces du gradient :

FIGURE 3 – Image des forces du gradient

1.2 Seuillage des images de gradients

Voici l'en-tête de notre fonction de seuillage de gradient :
function [image_binaire] = seuillage (gradient)

Cette fonction prend en entrée un gradient (G) et renvoie une image binaire (I_{bin}) avec seulement 0 pour un pixel noir et 1 (ou 255) pour un pixel blanc. Notre opération de seuillage est :

- Pour une image avec 3 canaux, si un seul pixel parmis les 3 canaux est superieur à 0.35 alors le pixel de sortie est un pixel blanc. Dans le cas contraire, il s'agit d'un pixel noir.

$G(x, y, R) > 0.35$ ou $G(x, y, G) > 0.35$ ou $G(x, y, B) > 0.35 \longrightarrow I_bin(x, y) = 1$

- Pour une image avec un cannal, si le pixel est superieur à 0.25 alors le pixel de sortie est un pixel blanc. Dans le cas contraire, il s'agit d'un pixel noir.

$$G(x, y) > 0.25 \longrightarrow I_{bin}(x, y) = 1$$

Nous explicitons les choix des paramètres de seuillage dans la prochaine section . Voici l'image obtenu après seuillage :

FIGURE 4 – Image des gradients après seuillage

1.3 Dilatation des arêtes trouvées

Voici l'en-tête de notre fonction de seuillage de gradient :
function [dilated_image] = dilate_image(image, N)

Cette fonction prend en entrée l'image binaire et N le paramètre de dilatation qui s'effectue dans un voisinage de taille NxN. L'opération de dilatation est tout simplement l'opération inverse de l'érosion et elle consiste à dilater , comme son nom l'indique, les pixels blancs (les arêtes). Le principe est de parcourir tous les pixels blancs de l'image et de recopier la valeur du pixel sur un voisinage de taille NxN (carré ou cercle) centré sur le pixel original.

Voici le principe de l'opération de dilatation :

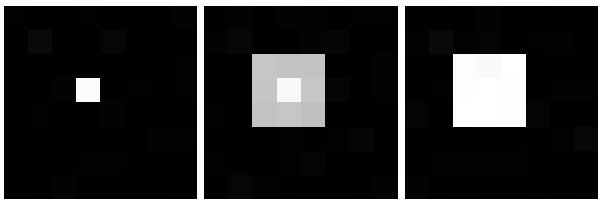


FIGURE 5 – Principe de dilatation sur un voisinage 3x3

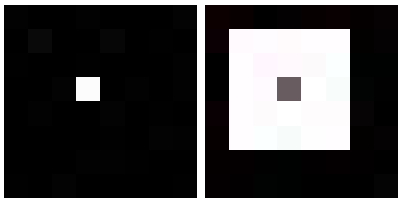


FIGURE 6 – Principe de dilatation sur un voisinage 5x5

Voici l'image obtenu après dilatation :

FIGURE 7 – Image avant dilatation (à gauche) et après dilatation (à droite)

2 Stratégie de seuillage

Les paramètres de seuillage est une étape importante pour tous les algorithmes impliquant un traitement sur les contours d'une image. En effet, un des objectifs principaux du seillage consiste à conserver seulement les contours en filtrant les bruits ou arêtes considéré comme inutile. Nous allons voir ainsi les deux approches obtenu lors de l'élaboration d'une stratégie de seuillage optimale.

2.1 1ère approche

Notre première approche naive consistait à fixer un seuil de 0.5 :

Pour une image RGB :
$G(x, y, R) > 0.5$ ou $G(x, y, G) > 0.5$ ou $G(x, y, B) > 0.5 \longrightarrow I_bin(x, y) = 1$

Pour une image noir et blanc :
$G(x, y) > 0.5 \longrightarrow I_bin(x, y) = 1$

Voici les images obtenus pour des images en couleurs :

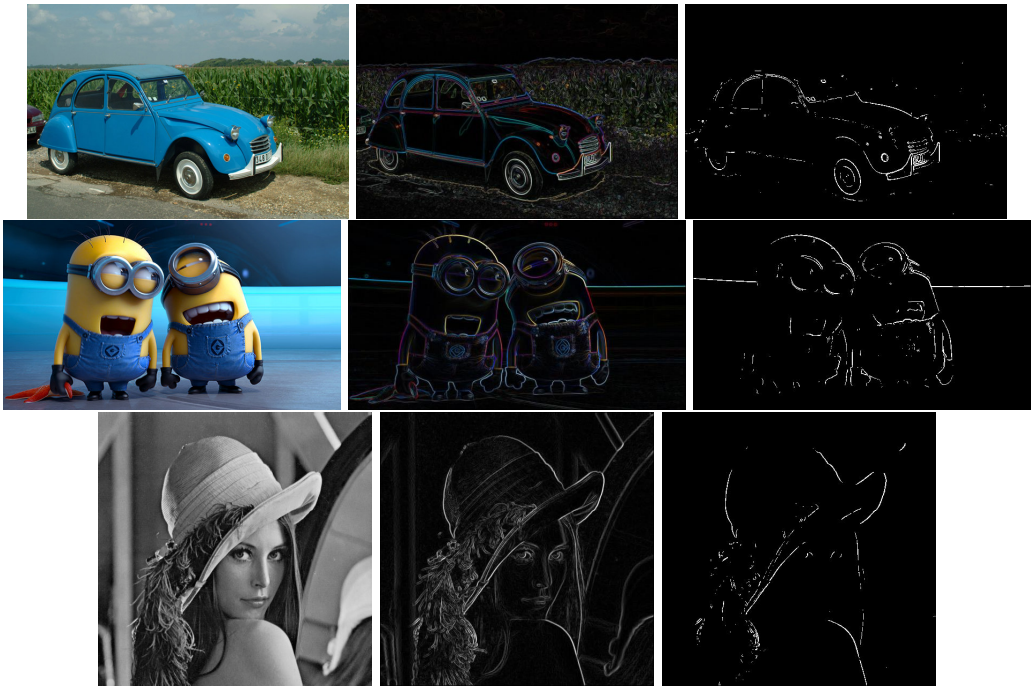


FIGURE 8 – Image originale (RGB) et image avant et après seuillage du gradient

Voici les images obtenus pour des images en noir et blanc :

FIGURE 9 – Image originale (noir et blanc) et image avant et après seuillage du gradient

Nous remarquons que le seuil de 0.5 est élevé pour les images RGB car la totalité du contours n'est pas conservé et le seuil est énormément élevé pour les images en noir et blanc car il filtre pratique plus de 75% du gradient de l'image (ex image de la rose). Cela dépend également de la résolution de l'image (ex image des minions). Nous devons donc procéder à une autre stratégie de seuillage.

2.2 2ème approche

Nous allons donc réfléchir à un seuil n'étant pas forcément optimale mais qui étant adaptable pour une large catégorie d'image que cela soit une image RGB ou noir et blanc mais également s'il s'agit d'une image à faible ou forte résolution. Toutefois avec notre seconde approche, pour ne pas obtenir la même image de gradient, nous considérons que si l'image a une trop faible résolution alors il est réellement impossible d'obtenir les contours globales.

Nous avons donc procéder à des tests pour différents seuils. Voici les images obtenus les images en couleur :

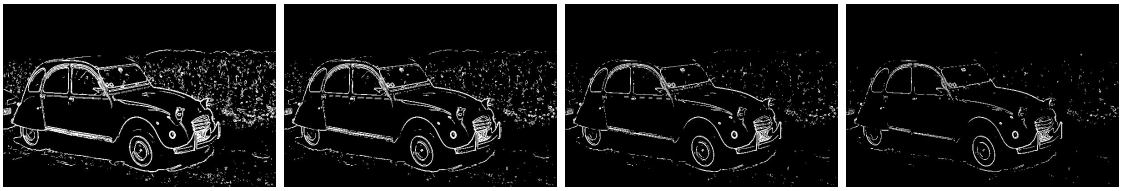


FIGURE 10 – Seuil respectivement 0.25 , 0.3, 0.35, 0.4

Voici les images obtenus les images en noir et blanc (faible résolution) :

FIGURE 11 – Seuil respectivement 0.25 , 0.3, 0.35

Nous remarquons que la résolution de l'image est très importante mais également une différence par rapport au image RGB qui comporte plus de pixels q'une image noir et blanc. Ce qui nous amené à differencier le seuil pour les images RGB et le seuil pour les images noir blanc. Voici les seuils que nous avons fixé d'après les tests effectués :

Pour une image RGB :
$G(x, y, R) > 0.3$ ou $G(x, y, G) > 0.3$ ou $G(x, y, B) > 0.3 \longrightarrow I_bin(x, y) = 1$

Pour une image noir et blanc :
$G(x, y) > 0.25 \longrightarrow I_bin(x, y) = 1$

Finalement, nous remarquons qu'il n'y a pas de stratégie de seuillage universelle car cela dépend des images. De plus, le seuillage est important pour les opérations post-seuillage comme la dilatation :

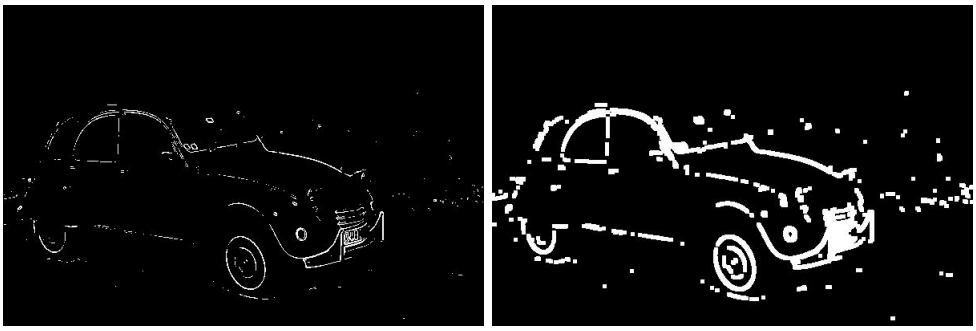


FIGURE 12 – Image avant dilatation (à gauche) et après dilatation (à droite) pour un seuil de 0.5

FIGURE 13 – Image avant dilatation (à gauche) et après dilatation (à droite) pour un seuil de 0.5

3 Stratégie de détection de transitions

3.1 Paramètre de seuillage

De manière générale, nous reprenons les mêmes paramètres de seuillage vus dans la section précédente pour cette étape de détection de transitions.

3.2 Paramètre de dilatation

Le paramètre de dilatation qui correspond à la taille N du voisinage NxN est également très important et figure comme une étape primordiale pour la détection des anciennes ou nouvelles arêtes. Le but de cette opération ici, permettra de comparer une image dilatée avec une image non dilatée. Cela revient à dire de comparer les arêtes d'une image avec une zone ou un voisinage des arêtes en question.

Le principe sera de comparer des arêtes entre des trames t et $t+k$ par le biais de la fonction *edge_ratio()*. Ceci permetra de detecter si une arète est en mouvement ou l'apparition d'une nouvelle arête. Il convient donc de ne pas choisir une taille de voisinage trop basse car cela ne permettra de reperer l'arête en mouvement qui peut se déplacer rapidement. De même, il ne faut pas choisir une taille de voisinage très élevée de manière à obtenir un gros pavé blanc. Ainsi, il peut avoir confusion entre les arêtes actuelles et les nouvelles arêtes.

Nous avons finalement choisi une taille de voisinage égale à 5.

3.3 1ère méthode

3.3.1 Principe

Le but de cette méthode est d'appliquer trame par trame les opérations vus précédemment, c'est à dire dans l'ordre :

1. *fg_norm_sobel()*
2. *seuillage()*
3. *dilate_image()*

Nous allons ensuite comparé les arêtes des trames t et $t+1$ par le biais de la fonction *edge_ratio()*. Ce qui nous donnera le ratio des arêtes entrantes (p_{in}) et le ratio des arêtes sortantes (p_{out}).

3.3.2 Détection de coupure

Nous savons que s'il y a une coupure alors il y a un changement des arêtes entrantes et des arêtes sortantes ce qui correspond à un très grand changement des ratios (p_{in}) et (p_{out}) avec un pic très élevé. En effet, nous devons retrouver à peu plus de 70% de diffèrences d'arêtes (cette valeur peut dépendre du paramètre de dilatation).

Nous prenons donc un seuil de 0.75 pour le $max(p_{in},p_{out})$ ce qui correspondra à une coupure.

3.3.3 Détection de fondus

Pour la détection de fondus, nous devons observer un léger pic pour p_{in} et p_{out} et également une variation de (p_{in}> p_{out}) puis de (p_{in}<p_{out}).

C'est pourquoi nous fixons un seuil pour début la détection de fondu à 0.15 pour le *max(p_{in},p_{out})* qui correspondra à un milieu de fondu et un deuxième seuil à 0.10 pour éviter de redétecter le même fondu. A partir de cela, il est possible de determiner approximativement le début et la fin du fondu mais nous avons decidé de detecter seulement le nombre de fondus car l'oeil humain ne detecte le début du fondu qu'après le réel début du fondu.

3.3.4 Modification de l'opération de convolution

Pour pouvoir détecter correctement le pic pour les fondus, nous avons été obligé de modifier l'opération de convolution. Nous n'appliquons pas la convolution sur les bords ou frontières de l'image. Ce qui revient à dire que nous démarrons i et j à 2 et nous terminons pour $nb_{lignes} - 1$ et respectivement $nb_{colonnes} - 1$ (sur Matlab).

3.4 2ème méthode

3.4.1 Principe

Le principe de cette méthode est la même que la première méthode sauf que nous comparons la trame *t* avec la trame *t+60*. L'objectif est de pouvoir détecter un fondu comme étant une coupure.

Nous aurons donc pour une coupure 60 trames ayant une valeur constante supérieur à 0.75 et nous aurons pour un fondu des valeurs supérieur à 0.75 mais variable pendant le fondu.

L'avantage de cette méthode est de determiner un fondu ou une coupure avec seulement un seuil. L'inconvénient est que nous obtenons un décalage de *t+60* et qu'il est difficile de déterminer avec précision le début ou la fin d'un fondu. et également s'il y a un fade in ou fade out.

3.4.2 Paramètre de seuil de coupure et fondus

Nous prenons donc un seuil de 0.75 pour le *max(p_{in},p_{out})* ce qui correspondra à une coupure s'il n'y a pas réel variation et correspondra à un fondu s'il y a une variation (augmentation puis diminution).

4 Présentation des résultats de détections pour 'TP3_video.avi'

4.1 1ère méthode

4.1.1 Résultats

Voici le graphique des p_{in} et p_{out} en fonction des trames obtenues :

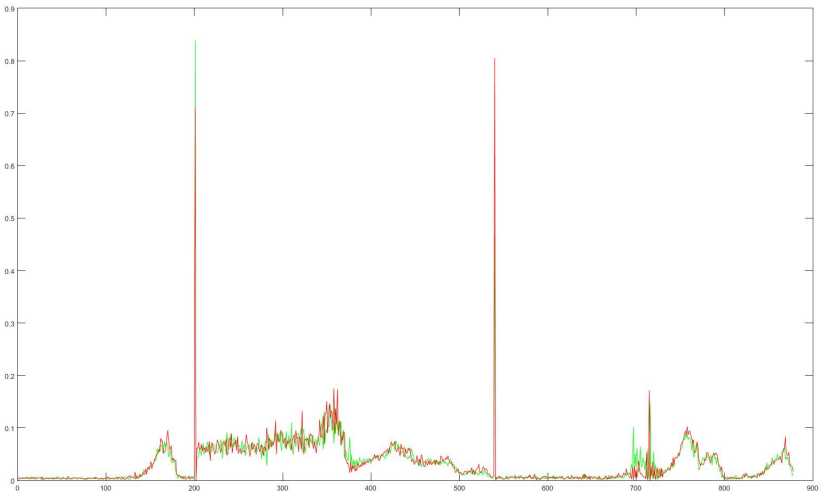


FIGURE 14 – Graphique des p_{in} et p_{out}

Notre algorithme nous donne les valeurs suivantes :
— $nb_{coupure} = 2$ avec $T_{coupure}$: 201 , 540
— $nb_{fondu} = 2$ avec T_{fondu} : 350 , 715

Nous sommes plutot satisfait des valeurs car pour les fondus, les réelles intervalles sont [345,395] et [675,735].

Voici le graphique obtenu si nous avions pas modifié l'opération de convolution comme convenue :

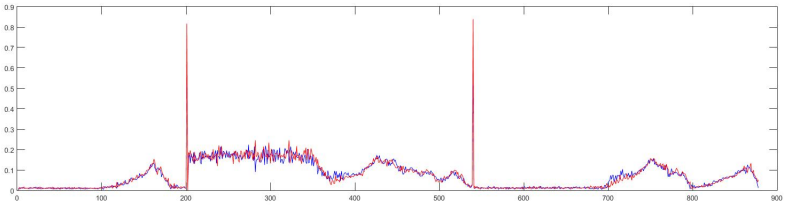


FIGURE 15 – Graphique des p_{in} et p_{out} (avant changement de *convolution()*)

4.1.2 Alternatives

Pour valider notre modèle et les valeurs des paramètres choisies, nous avons essayer avec d'autres valeurs. Voici les graphiques obtenues :

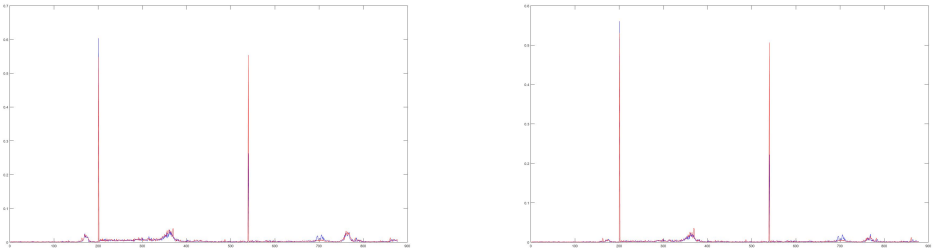


FIGURE 16 – Graphique des p_{in} et p_{out} (avec seuil=0.25 et N=9 à gauche et N=11 à droite)

La dilatation permet de detecter correctement les arêtes actuelles. Comme par exemple dans notre vidéo, la sortie du camion et la séquence du lac sont attenuées contrairement au premier graphique mais diminue fortement les seuils préfixés.

4.2 2ème méthode

4.2.1 Résultats

Voici le graphique des p_{in} et p_{out} en fonction des trames obtenues avec la seconde méthode :

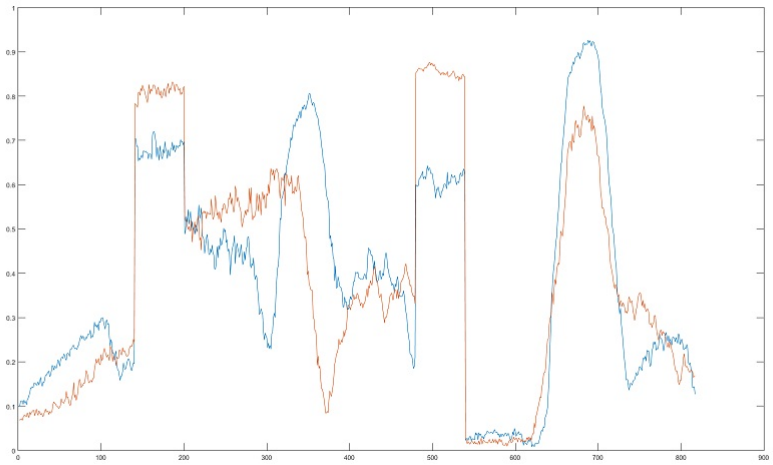


FIGURE 17 – Graphique des p_{in} et p_{out} avec la seconde méthode

Nous obtenons comme convenu les deux coupures et les deux fondus.

4.2.2 Alternatives

Voici le graphique obtenu avec un seuil de 0.25 et N=9 :

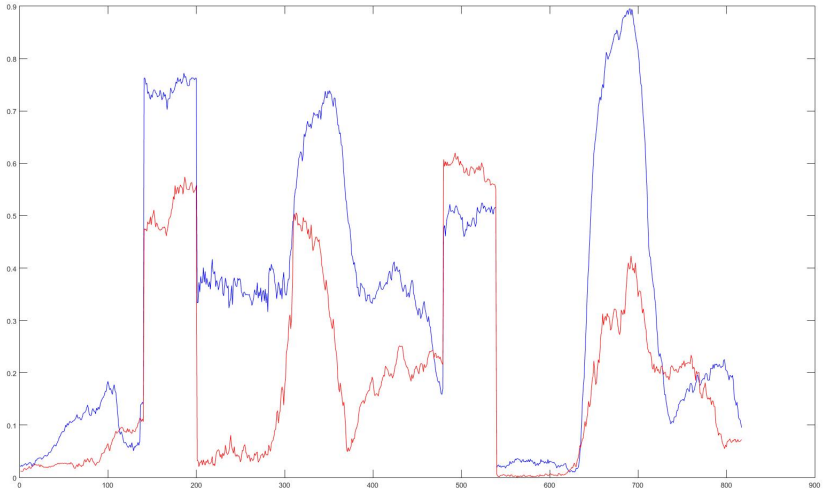


FIGURE 18 – Graphique des p_{in} et p_{out} avec la seconde méthode et seuil=0.25 et N=9

Certaines coupures sont atténuées et certains fondues sont amplifiées. Les variations semblement aléatoires. Il était également possible des trames t et *t+20* ou *t+30* :

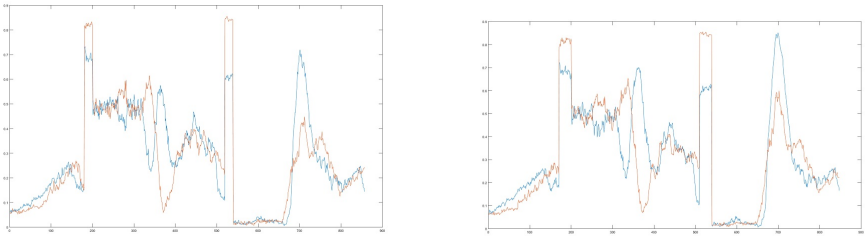


FIGURE 19 – Graphique des p_{in} et p_{out} avec la seconde méthode et $k = 20$ (à gauche) et $k = 30$ (à droite)

Nous remarquons que plus la valeur de k est petite, plus notre graphique va tendre vers le graphique initiale (avec $k = 1$) car les fondus sont moins détectés comme des coupures. Toutefois, plus la valeur de k est elevée, plus les valeurs de fondues seront très variable. En effet, nous avons choisi $k = 60$ car cela correspond à peu près au nombre de trames pendant un fondu.

5 Conclusion

Finalement, ce dernier TP nous aura permis de comprendre l'importance des paramètres lors de l'élaboration des stratégies de seuillage et de détection de transitions mais également de mettre en place les connaissances du cours pour pouvoir implémenter des fonctions à partir de rien (aucun fichiers sources).

Nous tenons également à signaler que lors des tests de détection de transition dans une séquence vidéo sous Matlab, nous avons remarqué une image (subliminale ?) qui apparaissait plusieurs fois très rapidement et qui disparaissait aussi rapidement. Il faut dire que cette image nous a perturbé pendant nos tests.

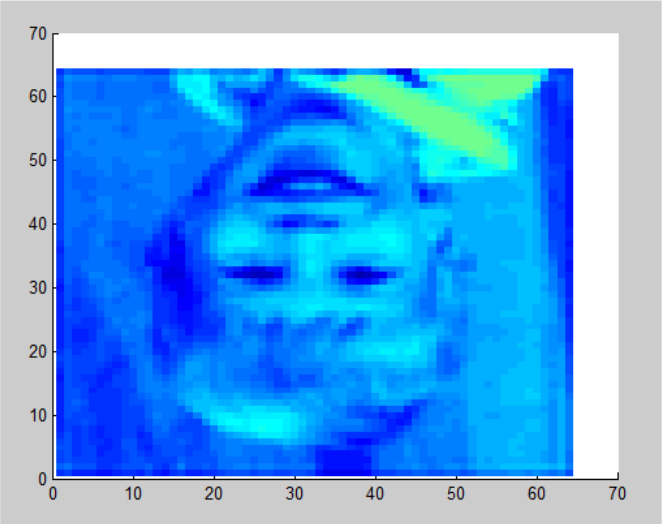


FIGURE 20 – Image bizarre de Matlab