



**Université
de Lille**
1 SCIENCES
ET TECHNOLOGIES

Université de Lille

MASTER 2 INFORMATIQUE
Premier semestre



**FACULTÉ
DES SCIENCES ET
TECHNOLOGIES**
Département Informatique

Vision artificielle

Fuzzy C-Means

BARCHID Sami

Année académique 2019-2020

Introduction

La matière vue dans ce TP traite de la segmentation d'image couleur par utilisation d'une technique de clustering en logique floue (« fuzzy c-means ») et ses variantes. Le but de ce TP est d'implémenter l'algorithme de Fuzzy C-means et plusieurs de ses variantes, puis d'analyser les résultats obtenus sur l'image ci-dessous.

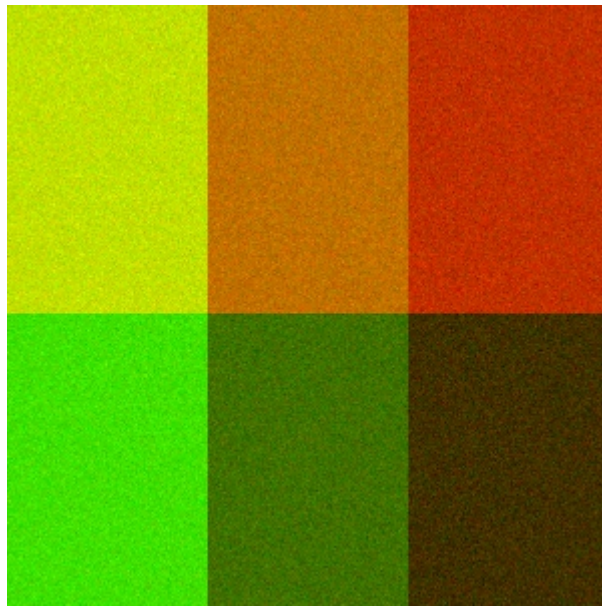


FIGURE 1 – Image de test bruitée

Ce rapport sera divisé avec les parties suivantes :

- **C-Means classique** : résumé rapide de l'algorithme C-Means classique, afin de rapidement comprendre les variantes en logique floue sur lesquelles on travaille.
- **Fuzzy C-means** : explication de fuzzy c-means et analyse des résultats obtenus par son utilisation sur l'image de test.
- **Hard C-means** : explication de hard C-means (c'est en réalité le C-means classique) et analyse des résultats obtenus par son utilisation sur l'image de test.

- **Possibilistic C-means** : explication du possibilistic C-mean et analyse des résultats obtenus par son utilisation sur l'image de test.
- **Algorithme de Davé** : explication de l'algorithme de Davé et analyse des résultats obtenus par son utilisation sur l'image de test.

C-means classique

L'algorithme C-Means est une méthode de clustering dont l'objectif est de classer les individus d'une population dans des partitions (ou « classes ») dont le nombre est défini par l'utilisateur. Ces individus présentent plusieurs caractéristiques qui vont permettre de faire cette répartition en classes. Dans notre expérience, la population est l'ensemble des pixels de l'image (donc un individu = un pixel) et les caractéristiques d'un individu sont les composantes - RGB d'un pixel.

Algorithme

L'algorithme C-means requiert qu'on lui fournisse un nombre de classes au préalable. Au départ, l'algorithme place le « centroïd » de chaque classe aléatoirement. Un centroïd est un échantillon dans l'espace des caractéristiques de la population qui va servir à assigner les individus selon une notion de distance. Un individu appartiendra à la classe pour laquelle le centroid est le plus proche (notion de distance à un centroïd, donc).

C-means est un algorithme itératif. À chaque itération, les centroids sont recalculés : les caractéristiques d'un centroïd sont la moyenne des caractéristiques des individus appartenant à la classe liée au centroïd. Après ce remplacement du centroïd, on réassigne les individus à la classe du nouveau centroïd le plus proche. On peut faire ça jusqu'à ce que la réassignation des individus devienne stable (on peut aussi définir d'autres conditions d'arrêt, comme le nombre d'itération, etc).

Le procédé de l'algorithme C-means est décrit ci-dessous.

- Placer les C centroïds aléatoirement.
- **TANT QUE** l'assignation des individus n'est pas stable OU que le nombre d'itérations maximal n'est pas dépassé
 - Assigner les individus au centroïd le plus proche.

- **POUR CHAQUE classe** : recalculer le centroïd (moyenne des caractéristiques des individus de la classe).

Un problème de C-means est le fait que l'algorithme est sensible au placement aléatoire initial des centroïds. Par exemple, si deux centroïds sont placés très proches l'un de l'autre au départ, ils se rapprocheront encore plus, divisant ainsi un regroupement d'individus en deux alors qu'ils représentent en réalité la même classe.

Un autre problème majeur de C-means est la nécessité de lui fournir le nombre de classes c de manière empirique. En effet, si on calcule un c trop grand ou trop petit, on n'obtiendrait pas une segmentation valable. Par exemple, dans notre image de test, on remarque rapidement qu'il y a 6 couleurs différentes donc 6 classes. Si on choisissait 3 classes par exemple, notre segmentation n'aurait pas de sens. La figure 2 montre une segmentation avec C-means classique en utilisant $c=3$, qui est donc un c trop petit.

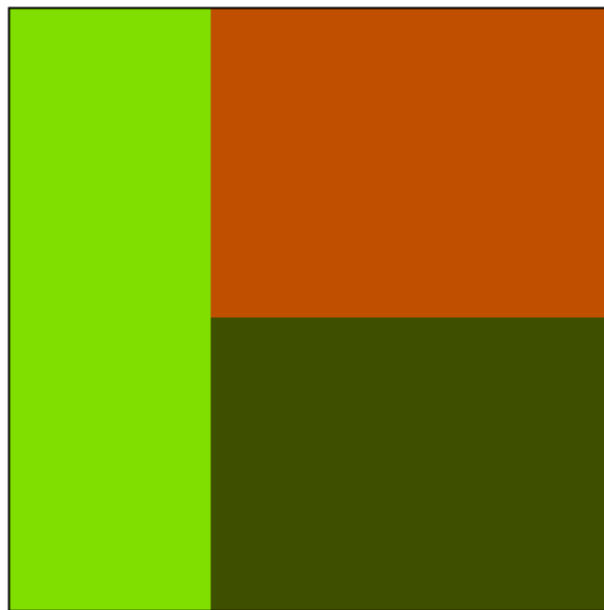


FIGURE 2 – c-means avec $c=3$, un nombre trop peu élevé.

Fuzzy C-means

Maintenant que nous comprenons C-means, nous pouvons comprendre une généralisation de C-means en logique floue : « Fuzzy C-Means ». Ce qui change est le fait que les partitions se comportent comme des ensembles flous où les pixels possèdent des degrés d'appartenance à chacun d'eux. À chaque itération, un pixel est segmenté avec l'ensemble flou pour lequel il possède le plus grande degré d'appartenance. Ces degrés d'appartenance sont calculés sur base d'une distance avec les centroïdes des ensembles flous, comme pour le C-means classique.

Le squelette de l'algorithme de C-means est toujours le même. À chaque fin d'itération, on calcule un index de performance. L'index de performance est une fonction qui calcule le coût de la répartition des pixels. Le but de l'algorithme est de minimiser l'index de performance. Si, entre deux itérations, l'index de performance remonte ou n'évolue plus assez vite (selon **un certain seuil**), on arrête l'algorithme.

Notations

- On note $P = \{A_1, A_2, \dots, A_c\}$, les c ensembles flous.
- On note $X = \{x_1, x_2, \dots, x_n\}$, l'ensemble des n pixels de l'image.
- On note $\mu_{A_i}(x_j)$, le degré d'appartenance du pixel x_j à l'ensemble flou A_i .

Comme dit plus haut, le squelette de l'algorithme C-means est toujours le même. Cependant, les spécificités sont situées aux formules permettant de :

- Calculer le placement d'un centroid.
- Calculer le degré d'appartenance d'un pixel à un des ensembles flous.
- Calculer l'index de performance.

Les notations ci-dessous sont les formules utilisées pour ces trois calculs.

Placement d'un centroïd

ν_i est le centroid de A_i . La formule des centroids est la suivante :

$$\forall i \in \{1, 2, \dots, c\}, \quad \nu_i = \frac{\sum_{j=1}^n [\mu_{A_i}(x_j)]^m \cdot x_j}{\sum_{j=1}^n [\mu_{A_i}(x_j)]^m} = \frac{\sum_{j=1}^n u_{ij}^m \cdot x_j}{\sum_{j=1}^n u_{ij}^m}$$

En d'autres termes, un centroïd est une moyenne pondérée **par un poids** $m > 1$ (**défini au départ**) des données influencées par leur degré d'appartenance à l'ensemble flou lié au centroïd.

Degré d'appartenance

μ_{ij} est le degré d'appartenance d'un pixel x_j à l'ensemble flou A_i et est donné par la formule :

$$u_{ij} = \left[\sum_{k=1}^c \left(\frac{d^2(x_j, \nu_i)}{d^2(x_j, \nu_k)} \right)^{\frac{2}{m-1}} \right]^{-1}$$

ATTENTION, en voyant cette formule, on voit que $m \neq 1$ sinon nous avons un problème de division par 0 !

Index de performance

$J_{FCM}(P)$ est l'index de performance calculé après l'obtention de la partition floue P à la fin d'une itération. Sa formule est la suivante :

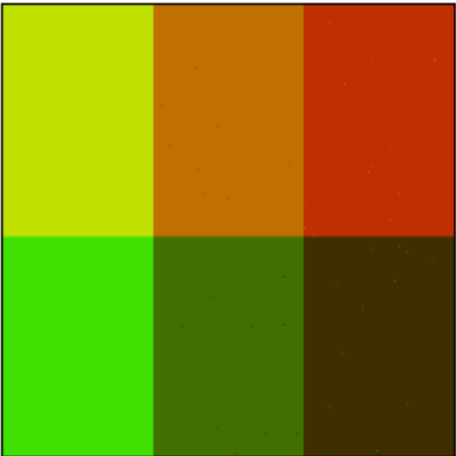
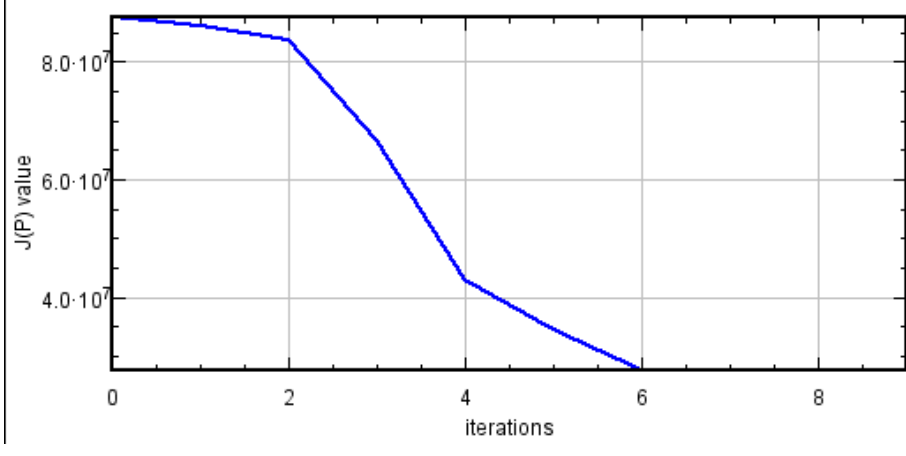
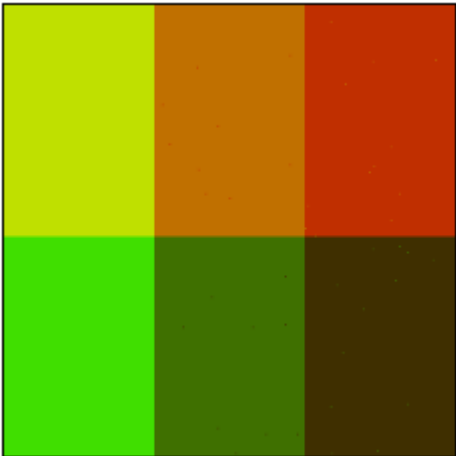
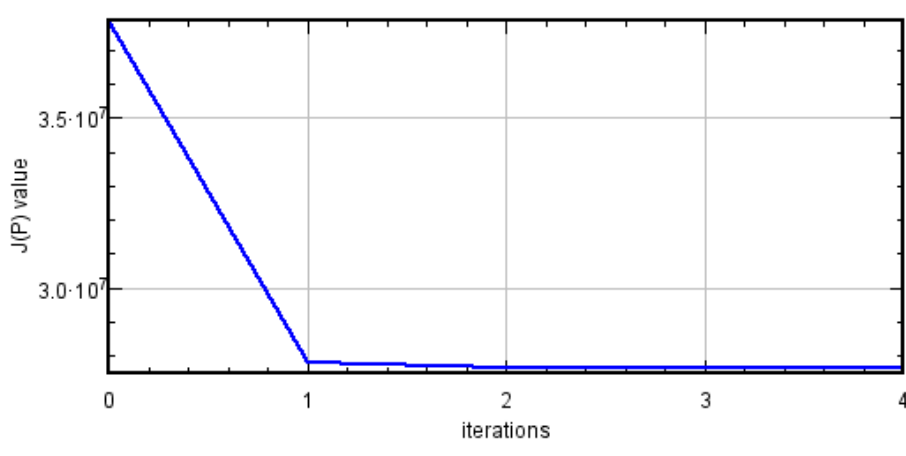
$$J_{FCM}(P) = \sum_{i=1}^c \sum_{j=1}^n [\mu_{A_i}(x_j)]^m \|x_j - \nu_i\|^2 = \sum_{i=1}^c \sum_{j=1}^n [u_{ij}]^m \cdot d_{ij}^2$$

Expérimentations

Pour cette expérimentation, on fait varier les paramètres d'entrée de l'algorithme suivants :

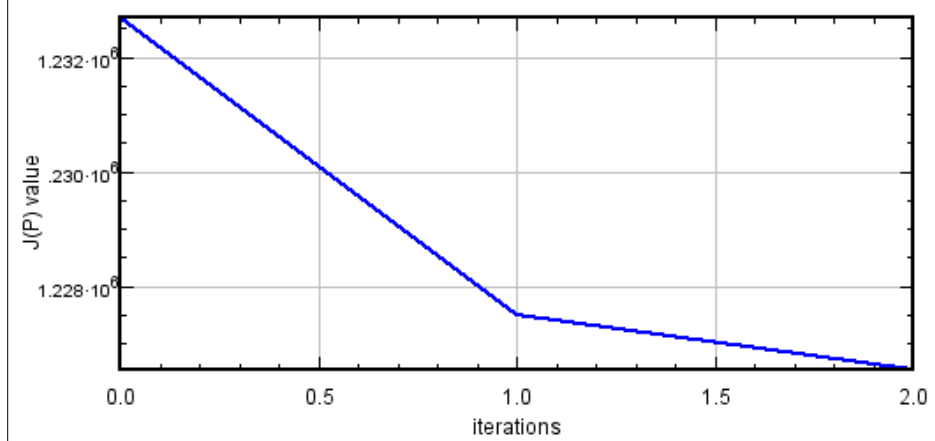
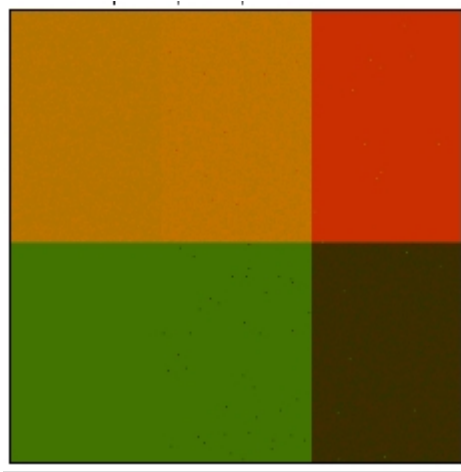
- m , le poids qui influence le degré d'appartenance
- Le seuil de stabilisation de l'algorithme. Si la différence entre l'index de performance de l'itération courante et celle d'avant est inférieure à ce seuil, alors on estime que l'algorithme est stable et on arrête l'algorithme.

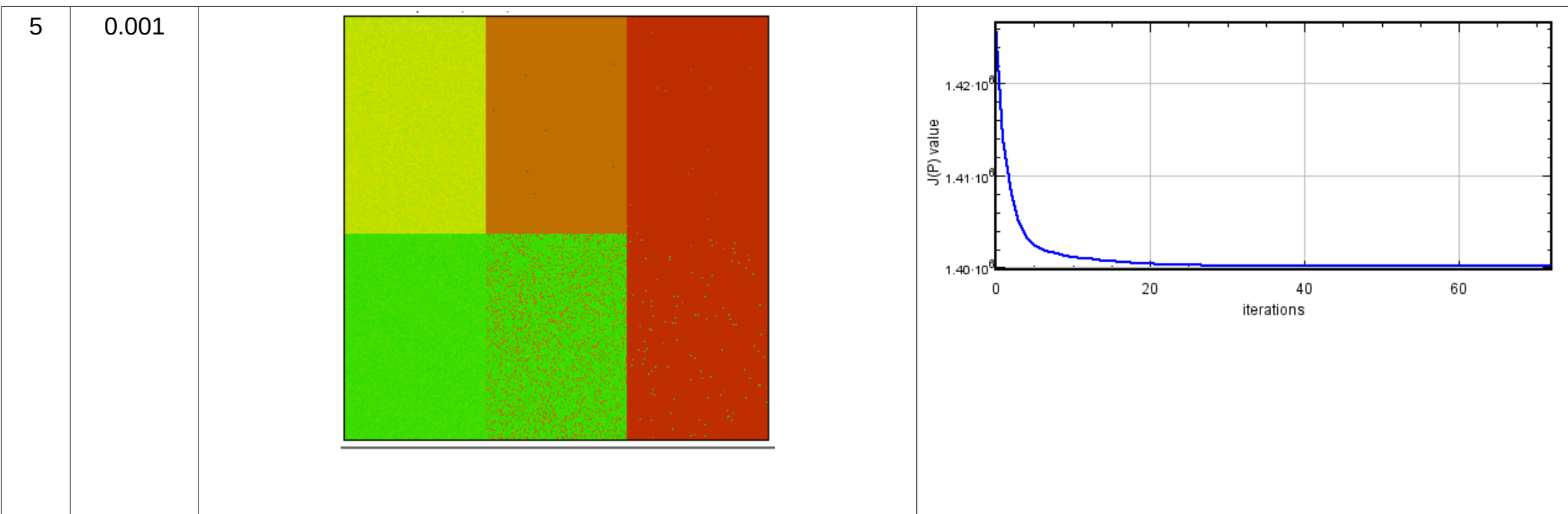
Chaque mesure prise exposera l'image segmentée obtenue et un graphique d'évolution de l'index de performance à chaque itération. On considère que le nombre d'itérations maximum choisi est assez grand pour laisser à l'algorithme le temps de se stabiliser. Le nombre de classes de l'image est $c=6$. Le tableau ci-dessous reprend les expériences menées :

m	Seuil stabilisation	Image segmentée obtenue	Évolution index de performance
2	0.001		
2	1.9		

5

1





On peut faire les observations suivantes :

- Utiliser un seuil plus petit permet d'augmenter le nombre d'itérations, ce qui permet d'apporter, normalement, une segmentation plus précise (mais qui dure largement plus longtemps).
- Un m trop grand a pour effet de brouiller le résultat et de faire mettre plus de temps à l'algorithme pour converger. Ceci peut s'expliquer par le fait que, pour le calcul du centroïd, plus un m est grand et plus les pixels qui sont dans la classe du centroïd auront de l'influence par rapport aux autres. Ça a pour effet qu'un centroïd devra mettre plus de temps à se replacer.

Hard C-means

Hard C-Means est en réalité l'algorithme C-Means classique, sauf que l'on utilise la généralisation floue Fuzzy C-Means pour le faire. Par rapport à FCM, plusieurs choses changent :

- $m=1$ dans tous les cas.
- Le calcul du degré d'appartenance à une classe a changé :

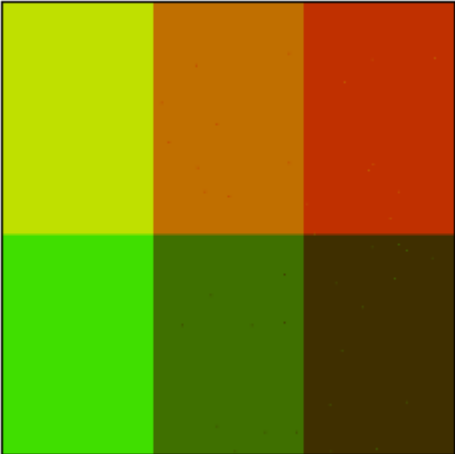
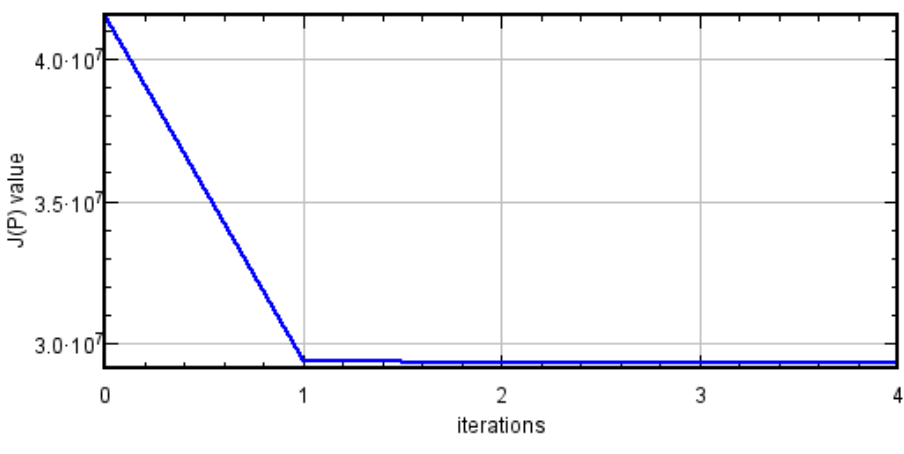
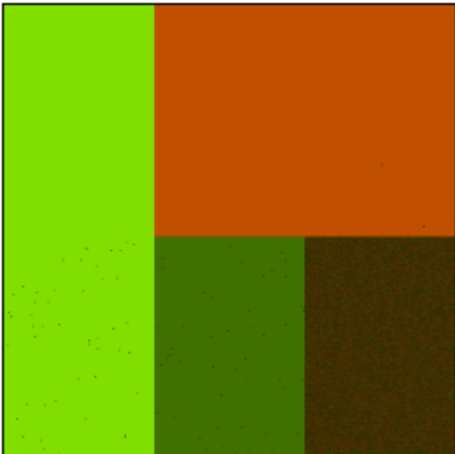
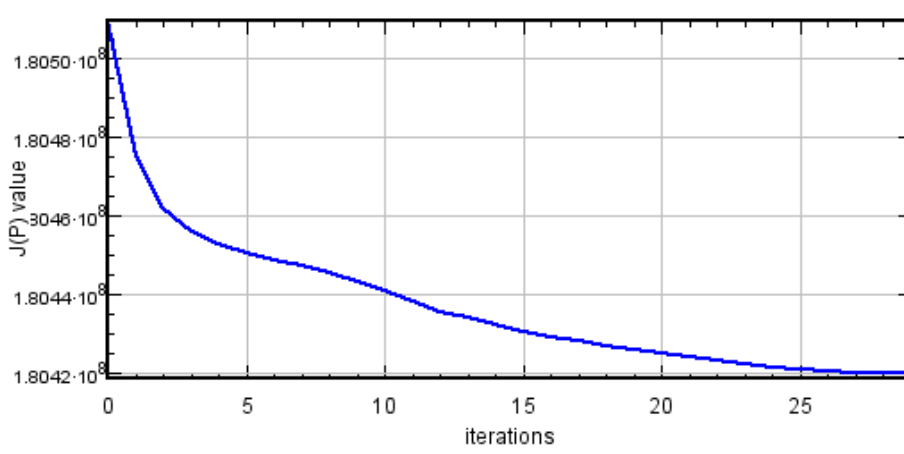
$$\forall i \in \{1, 2, \dots, c\}, \forall j \in \{1, 2, \dots, n\} \quad u_{ij} = \begin{cases} 1 & \text{if } d^2(x_j, \nu_i) < d^2(x_j, \nu_k) \quad \forall k \neq i \\ 0 & \text{otherwise} \end{cases}$$

Ces deux changements font que FCM devient le c-means classique :

- Le degré d'appartenance devient un assignment qu'on pourrait qualifier de « booléen » : on appartient totalement à la classe dont le centroïd est le plus proche et on n'appartient à personne d'autre, comme c-means classique.
- Le fait que $m=1$ dans tous les cas a pour conséquence que le calcul du centroid, par exemple, devient le calcul de la moyenne des données appartenant à la classe de ce centroid.

Expérimentations

Dans cette variante, on ne peut faire varier que le seuil de stabilisation.

Seuil stabilisation	Image segmentée obtenue	Évolution index de performance
1		
0.001		

Note : dans cette expérimentation, il faut noter que, pour l'expérience avec un seuil = 0.001, le placement aléatoire initial des centroïdes a été mal réglé, ce qui entraîné une segmentation fausse (d'où l'importance du placement au hasard de départ). On peut aussi mentionner le fait que l'expérience pour seuil = 1 converge très vite dû à un placement des centroïdes initial assez chanceux.

On peut remarquer les mêmes choses que pour la partie d'avant : un seuil réduit permet à l'algorithme de faire plus de corrections mais augmente le temps de calcul.

Possibilistic C-means

Possibilistic C-Means (PCM) est une variante du FCM qui a pour objectif d'être plus résistant au bruit. Les changements par rapport à FCM sont les suivants :

Calcul du degré d'appartenance

On introduit, pour chaque classe A_i , une distance η_i donné par la formule suivante :

$$\eta_i = \frac{\sum_{j=1}^n u_{ij}^m \cdot d_{ij}^2}{\sum_{j=1}^n u_{ij}^m}$$

Avec cela, on peut calculer le degré d'appartenance :

$$u_{ij} = \frac{1}{1 + \left(\frac{d^2(x_j, \nu_i)}{\eta_i} \right)^{\frac{1}{m-1}}} = \frac{1}{1 + \left(\frac{d_{ij}^2}{\eta_i} \right)^{\frac{1}{m-1}}}$$

On remarque que, contrairement aux autres algorithmes, μ_{ij} ne dépend que de la distance entre le pixel x_j et le centroïde ν_i . On ne calcule donc pas un degré relatif aux autres classes de la partition. Cela permet de détecter et corriger des pixels aberrants.

Index de performance

L'index de performance change pour être adapté à cette nouvelle définition du degré d'appartenance.

$$J_{PCM}(P) = \sum_{i=1}^c \sum_{j=1}^n [u_{ij}]^m \cdot d_{ij}^2 + \sum_{i=1}^c \eta_i \sum_{j=1}^n [1 - u_{ij}]^m$$

En réalité, un terme de pénalité a été ajouté à la formule de l'index de performance de base afin d'éviter le cas de figure où $\mu_{ij}=0 \forall i$ and $\forall j$.

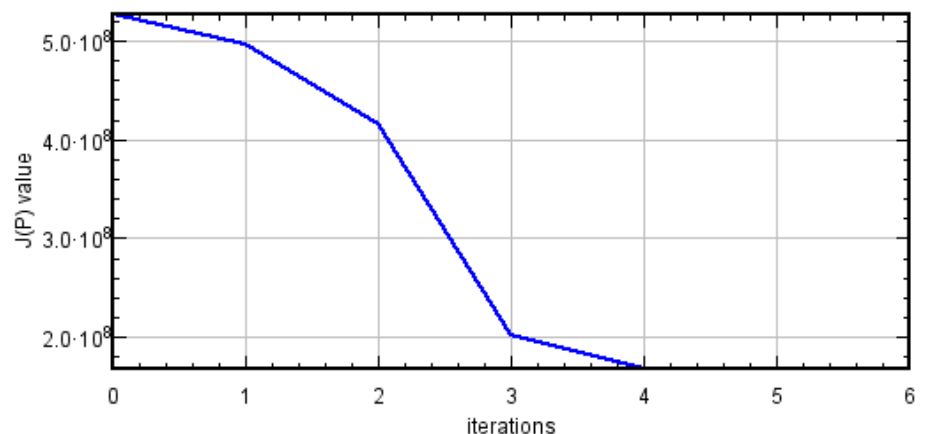
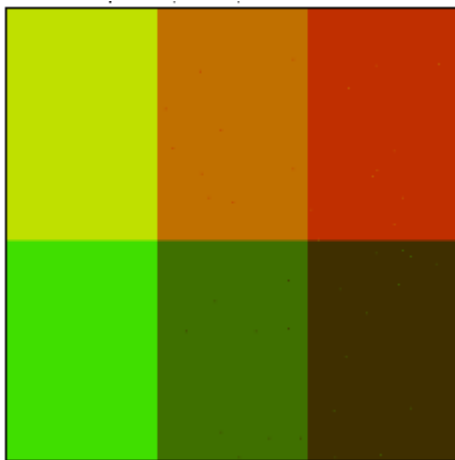
Expérimentations

Les expériences se feront en deux temps :

- Tester simplement l'algorithme pour voir s'il fonctionne
- Comparer la gestion du bruit avec le FCM classique

Test de PCM

On cherche à tester simplement si l'algorithme fonctionne et parvient à réaliser une segmentation correcte. Pour ce faire, on teste avec un $m=2$ et un seuil de 0.001. L'image ci-dessous est le résultat de l'expérience.



On observe que l'algorithme converge rapidement (5 itérations à peine) et les résultats sont correctes.

Gestion du bruit

En théorie, PCM est plus efficace pour gérer les pixels de bruits par rapport à FCM. On cherche donc à voir cette efficacité en comparant PCM et FCM sur une image de test modifiée avec encore plus de bruit (voir Figure 3).

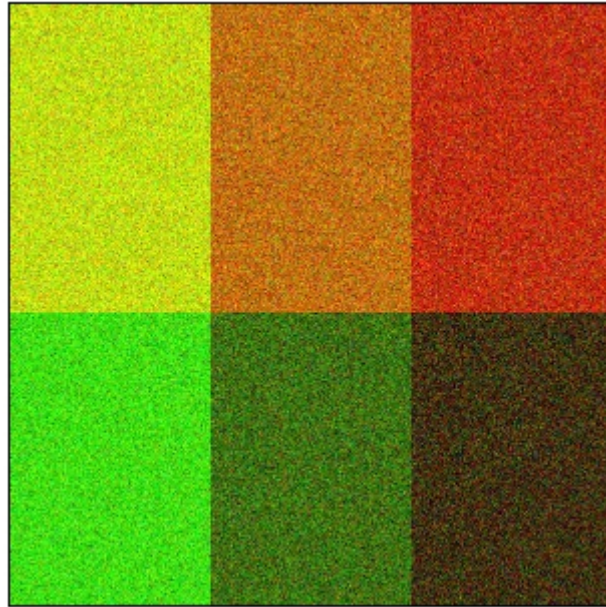
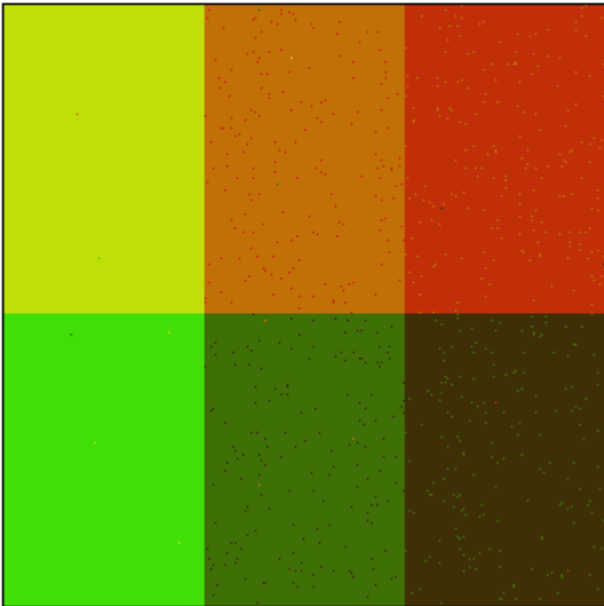
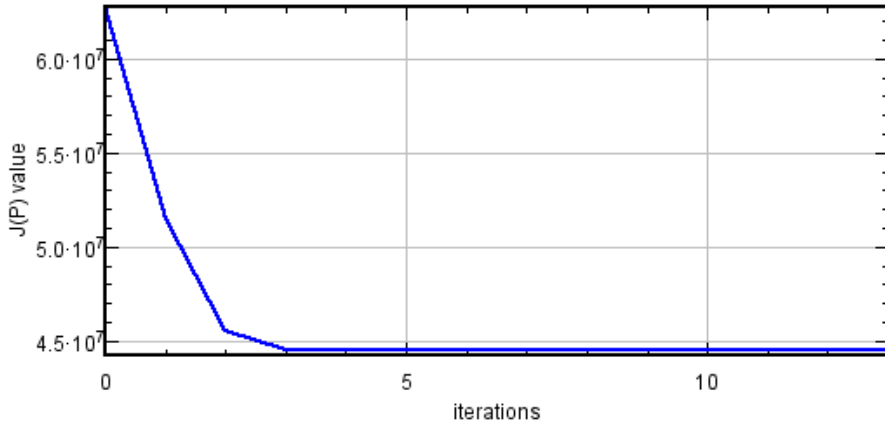
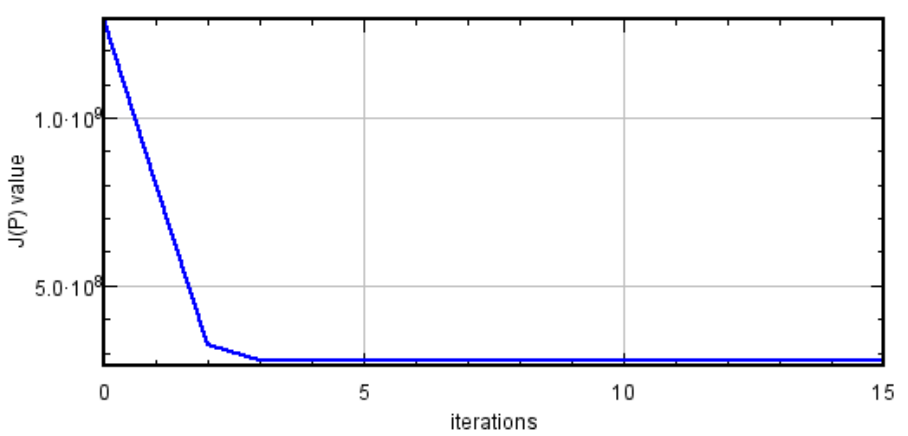
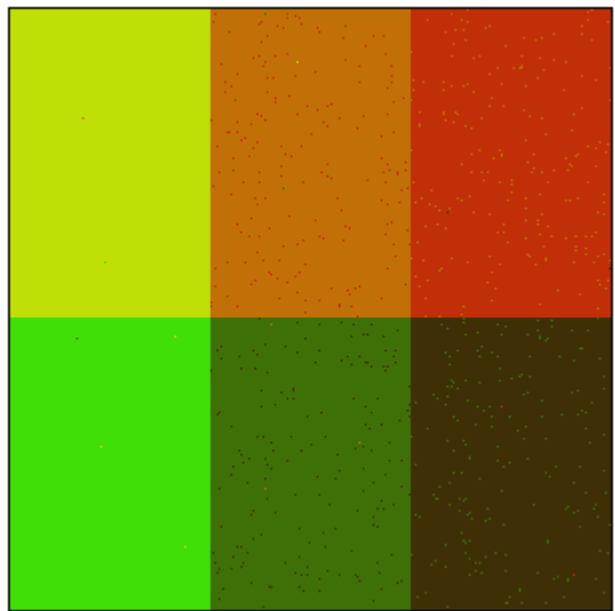


FIGURE 3 – image de test sur laquelle on a rajouté un bruit gaussien.

Le tableau ci-dessous compare les deux algorithmes :

Algorithm																										
Fuzzy C-means		 <table><caption>Approximate data points from the J(P) value vs iterations graph</caption><tr><th>Iterations</th><th>J(P) value ($\times 10^7$)</th></tr><tr><td>0</td><td>6.2</td></tr><tr><td>1</td><td>5.2</td></tr><tr><td>2</td><td>4.6</td></tr><tr><td>3</td><td>4.45</td></tr><tr><td>4</td><td>4.45</td></tr><tr><td>5</td><td>4.45</td></tr><tr><td>6</td><td>4.45</td></tr><tr><td>7</td><td>4.45</td></tr><tr><td>8</td><td>4.45</td></tr><tr><td>9</td><td>4.45</td></tr><tr><td>10</td><td>4.45</td></tr></table>	Iterations	J(P) value ($\times 10^7$)	0	6.2	1	5.2	2	4.6	3	4.45	4	4.45	5	4.45	6	4.45	7	4.45	8	4.45	9	4.45	10	4.45
Iterations	J(P) value ($\times 10^7$)																									
0	6.2																									
1	5.2																									
2	4.6																									
3	4.45																									
4	4.45																									
5	4.45																									
6	4.45																									
7	4.45																									
8	4.45																									
9	4.45																									
10	4.45																									

Probabilistic C-Means



On remarque ici que l'amélioration n'est pas flagrante. On émet l'hypothèse que le PCM permet de faire face à des bruits qui montrent des pixels aberrants plus marginaux que pour un bruit gaussien.

Algorithme de Davé

L'algorithme de Davé est une variante de FCM qui introduit une nouvelle classe en plus de celles d'avant : la classe « bruit ». Cette classe sert à détecter et segmenter les pixels aberrants (habituellement nommés « outliers »). Ce qui change par rapport à FCM est :

- Le degré d'appartenance à la classe bruit pour un pixel x_j est :

$$u_{\star j} = 1 - \sum_{i=1}^c u_{ij}$$

- L'index de performance doit prendre en compte cette classe « bruit » :

$$J_{Dav}(P) = \sum_{i=1}^c \sum_{j=1}^n [u_{ij}]^m \|x_j - \nu_i\|^2 + \sum_{j=1}^n \delta^2 \left(1 - \sum_{i=1}^c u_{ij}\right)^m$$

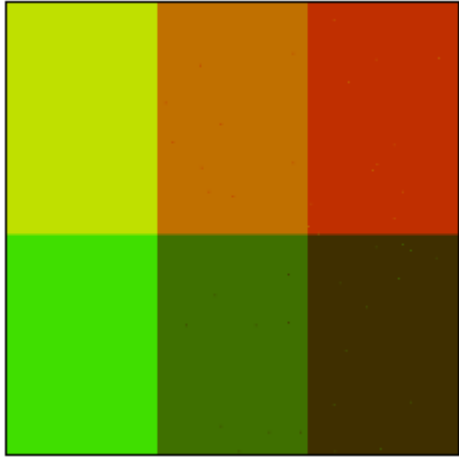
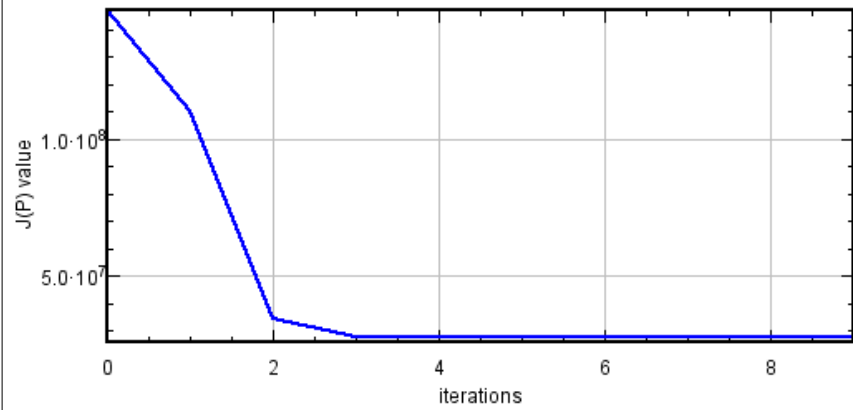
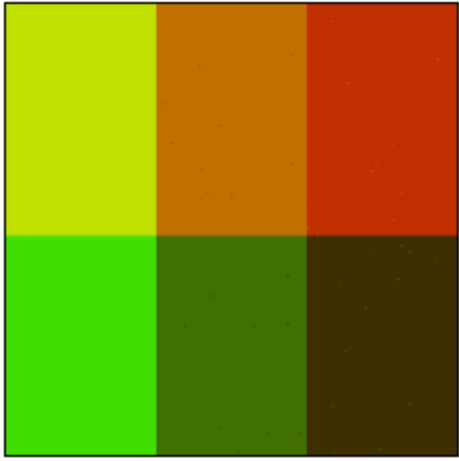
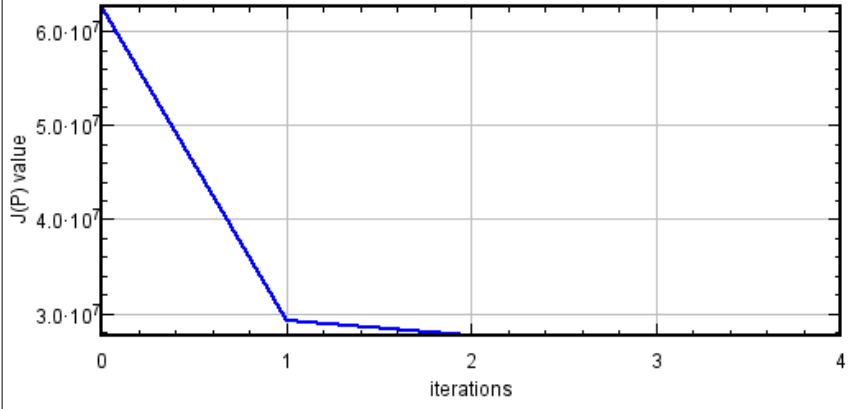
où δ^2 est une distance fixe de la classe bruit par rapport à toutes les autres classes. Cette distance permet de contrôler la quantité d'outliers tolérée par l'algorithme. Sa formule est la suivante :

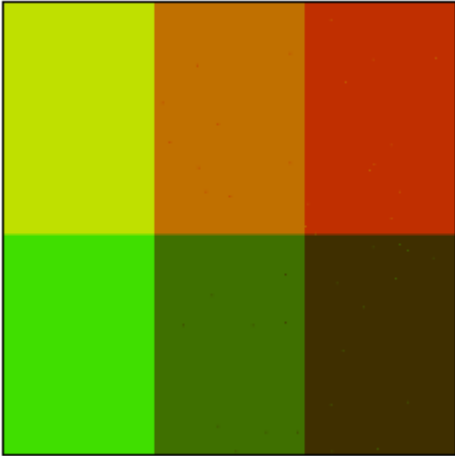
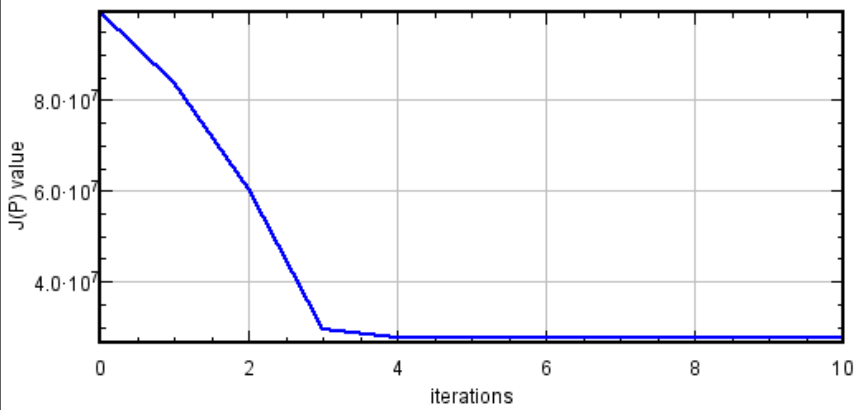
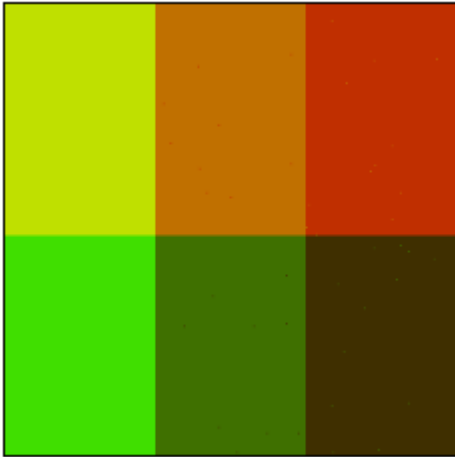
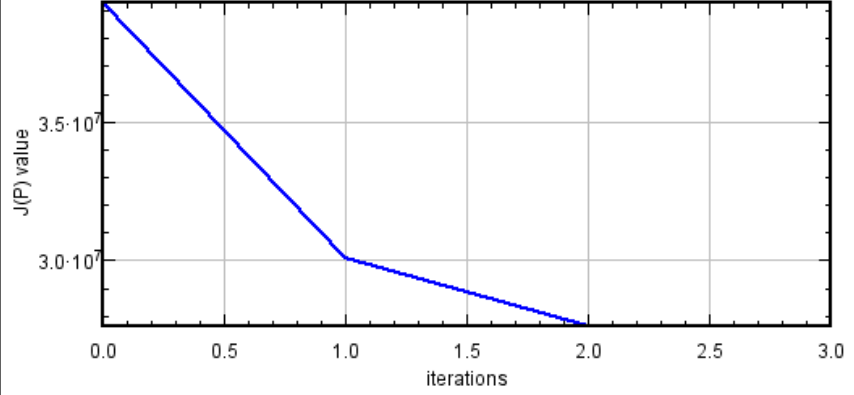
$$\delta^2 = \lambda \cdot \frac{\sum_{i=1}^c \sum_{j=1}^n [d_{ij}]^2}{n \cdot c}$$

avec λ qui est un nouveau paramètre à fournir au début de l'algorithme.

Expérimentations

On fait varier le m et le λ de l'algorithme. Les pixels « outliers » seront représentés en noir dans les images résultantes. Le tableau ci-dessous reprend les expériences :

m	λ	Image segmentée obtenue	Évolution index de performance
2	1		
2	10		

2	0.1		
2	100		

On remarque que, comme dans les slides présentant l'algorithme de Davé, il n'y a pas de pixels « outliers » marqués pour l'image de test. Ceci est dû au fait que l'image est trop simple et pas assez bruitée pour que l'algorithme puisse y trouver des outliers. On remarque aussi que l'algorithme fournit tout de même des résultats correctes.

Conclusion

Nous avons vu, dans ce TP, différentes variantes de l'algorithme C-means appliqué à la logique floue. Nous remarquons qu'en générale, toutes ces techniques fonctionnent relativement bien, avec leurs spécificités selon le cas d'utilisation (exemple : pour identifier les pixels aberrants, on pourrait plutôt choisir l'algorithme de Davé).

Nous pouvons néanmoins reconnaître que la segmentation offerte par cet algorithme ne fonctionne pas de manière totalement autonome puisqu'il faut tout de même fournir des paramètres à l'algorithme pour que celui-ci fonctionne correctement, comme le nombre de classes existantes, par exemple.

De plus, le problème du placement initial aléatoire des centroïdes peut être réellement handicapant et mène très souvent à des segmentations totalement fausses, comme le montrent les images ci-dessous.

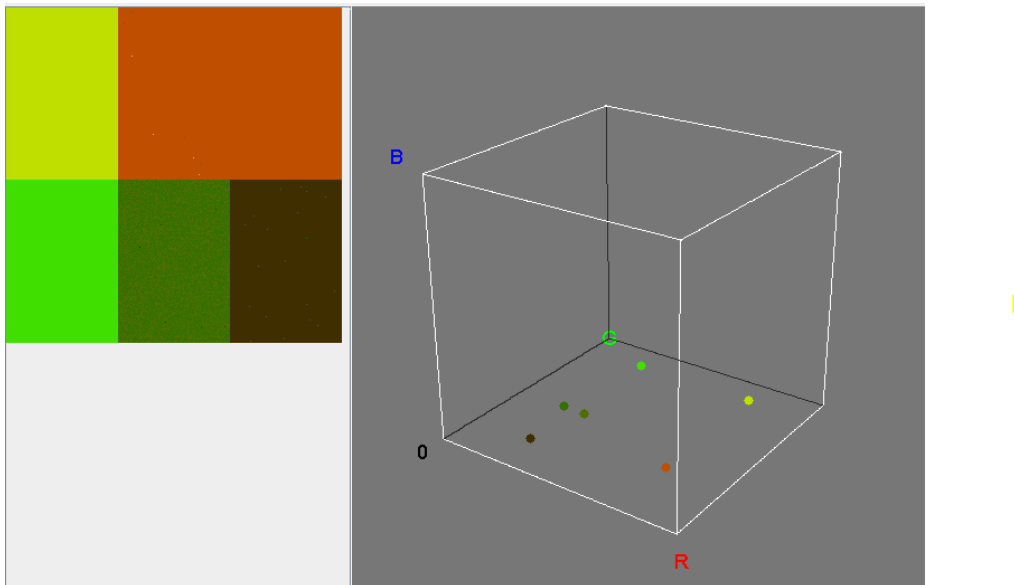


FIGURE 4 – segmentation fausse due à un placement initial aléatoire problématique. On voit dans la visionneuse de couleurs que deux centroïdes ont été placés trop proches l'un de l'autre (vert foncé).

Enfin, nous pouvons conclure que la segmentation offerte par c-means et ses variantes est efficace dans des cas où les images sont très simples, avec des couleurs peu nuancées. En revanche, on ne peut pas réellement parler de « segmentation » étant donné qu'on ne prend aucunement en compte la structure géométrique des pixels. On ne se soucie que de leurs couleurs et pas des formes qu'ils créent. En définitive, C-means ne pourrait pas être capable de reconnaître des entités telles que des objets dans des images naturelles.