

BINARISATION DE DOCUMENTS ANCIENS

El AISSAOUI Mohamed

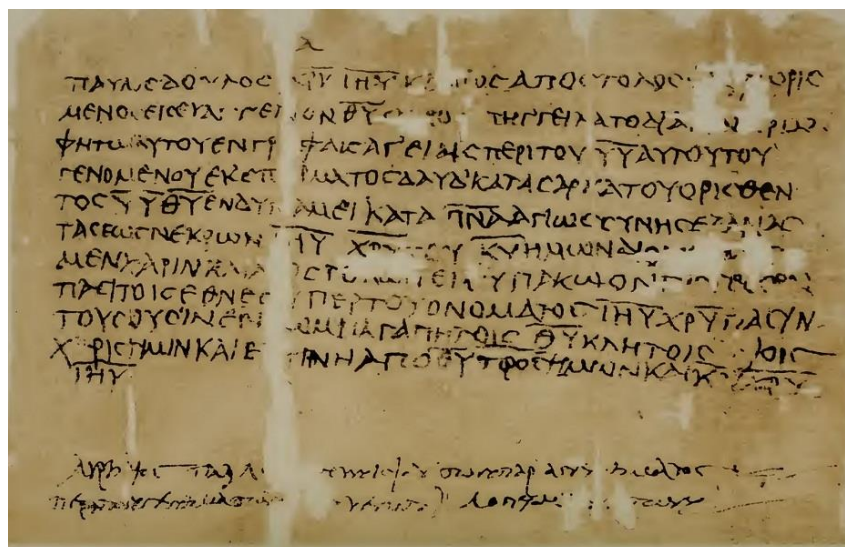


Table des matières

1. Introduction

2. Méthode Otsu

3. Méthode Canny

3.1 Réduction du bruit

3.2 Calcul de l'intensité du gradient

3.3 Suppression des valeurs non maximales

3.4 Seuillage double

3.5 Numérotation des objets

3.6 Seuillage

3.7 Amélioration de l'image

3.8 Elimination du bruit persistant

4. Méthode Sauvola

4.1 Formule de binarisation

4.2 Principe de la méthode

4.3 Application sans prétraitement

4.4 Application avec prétraitement

5. Conclusion

6. Bibliographie

7. Annexes

1 Introduction

Les manuscrits médiévaux sont des objets qui sont fortement endommagés et souvent combinés à un alphabet très varié. Afin d'exploiter ces documents altérés par le temps, des chercheurs ont mis en place des outils numériques afin d'extraire des données des images de manuscrits médiévaux de façon partiellement automatique. Afin de simplifier les traitements, de nombreuses recherches se sont orientées vers la binarisation de l'image.

Binariser une image revient à segmenter l'image en deux classes: le fond et l'objet. Il existe plusieurs types de méthodes de binarisation : Otsu, Sauvola...

En raison de la multitude de sources de bruits, et surtout de la multitude d'effets de ces bruits sur une image, il n'existe pas de technique de restauration générale, adaptée à toutes les situations. Il est donc important de savoir identifier la dégradation dont est victime l'image, afin d'appliquer le traitement adapté. Il existe deux types de dégradations les perturbations aléatoires (bruit) et les perturbations déterministes (comme le flou). L'objectif de cette étude est de trouver une méthode de binarisation la plus performante qui permet de distinguer de façon nette les écritures présentes sur le document de base. Le second consistera en la comparaison de cette méthode avec une méthode théorique plus complexe qui est la méthode de Sauvola.

2 Méthode Otsu

Durant la première séance, plusieurs recherches ont été effectuées afin de nous renseigner et nous familiariser avec plusieurs méthodes.

La première méthode que nous avons testée est celle d'Otsu. Cette méthode consiste en la recherche d'un seuil optimal dépendant de l'image qui sépare le mieux les composantes claires et foncé. Plus précisément, Otsu classe les pixels en deux catégories:

- Le fond : en dessous du seuil
- Les objets : au-dessus du seuil

Cette méthode cherche à minimiser la variance intra-classe et l'écart quadratique moyen entre la valeur des pixels et la moyenne de la classe à laquelle ils appartiennent.

Après avoir effectué le programme correspondant à cette méthode (cf. annexe) nous l'avons appliqué à la photo nommée 'H01' (fig.a).

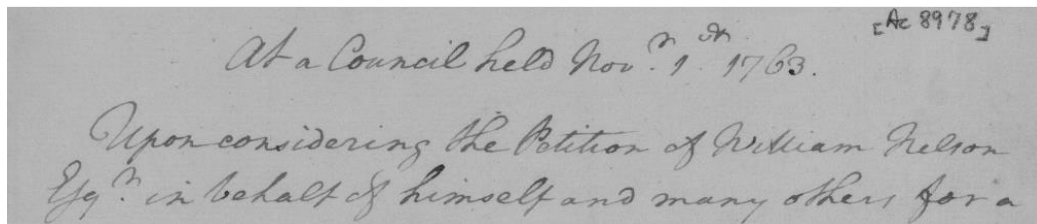


Fig.a Image d'origine H0

Nous obtenons le résultat suivant :

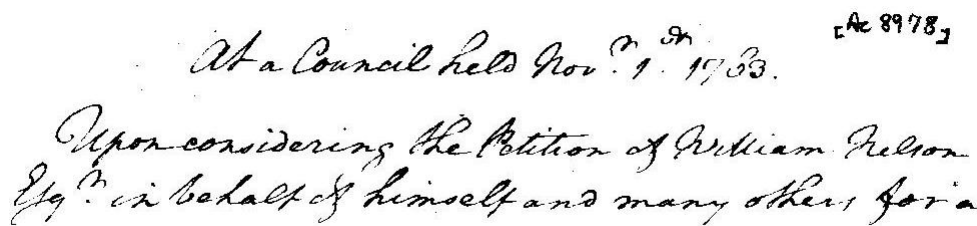


Fig.b Image Otsu

Nous observons que le fond gris est devenu blanc. Ce qui s'explique par le fait que les pixels du fond gris ont une intensité inférieure au seuil calculé par Otsu.

De même nous remarquons, que les 'taches noirs' présentes sur la photo d'origine sont toujours présentes dans l'image traitée. Ce seuillage par Otsu a également provoqué des pertes d'informations notamment au niveau du '7' présent sur la date '1763', ainsi que sur les lettres, en particulier les 'f'.

Ces pertes sont causées par le fait que les pixels qui définissent les objets ont une intensité inférieure au seuil, et ainsi sont confondus avec le fond de l'image.

En conclusion, l'application de cette méthode ne semble pas la plus adaptée pour binariser un document ancien. Pour cela, nous avons créé un autre programme visant à éliminer les taches et à éviter au maximum les pertes d'informations. Ce programme consiste à utiliser le filtre de Canny.

3 Méthode Canny

Le désavantage dans notre image est la présence dans le texte de pixels d'intensités trop faibles.

En effet, la complication dans la méthode de seuillage globale est la présence dans le texte de pixels avec un spectre d'intensités très large, si bien que la détermination d'une valeur seuil n'est pas évidente si l'on souhaite détacher le texte du fond tout en évitant un trop fort bruit.

Une méthode qui permet d'accentuer la reconnaissance de ces points souvent présents sur les bords des objets de l'image (par exemple sur le « 7 ») est la détection des contours. Par la suite afin de retrouver tout le texte on adjoindra les pixels de plus forte intensité.

La méthode que nous choisirons est celle de Canny : elle est plus complexe que d'autres méthodes mais présente l'avantage d'allier une bonne détection pour de faibles contours, une bonne localisation et d'avoir une seule réponse par contour.

3.1 Réduction du bruit

Avant de détecter les contours il est nécessaire de réduire le bruit qui pourrait générer un fort gradient causant la détection de lignes ne faisant pas partie du contour souhaité. Pour cela on applique un lissage, c'est à dire qu'on convolue l'image par un filtre. Numériquement le filtre est constitué par une matrice (masque) dont les éléments sont les coefficients du filtre. Il est placé sur chaque pixel de l'image. La valeur du pixel est calculée en faisant la somme pondérée des pixels de son voisinage.

Canny incorpore un filtre gaussien dont l'opérateur de convolution est :

$$G(x, y) = e^{\frac{-x^2 - y^2}{2\sigma^2}}$$

Son masque 5*5 ici, est :

$$\frac{1}{115} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Canny permet de définir l'écart-type du filtre (sigma). Sigma caractérise la largeur du filtre : en partant du point central la largeur est égale à 3sigma. La difficulté est d'établir un compromis entre réduction du bruit et conservation des détails. Dans notre cas, en testant différentes dimensions du filtre nous avons choisi un masque 5*5 qui permet

d'enlever du bruit sans perte significative d'information. Nous avons alors assigné $\sigma=0.7$ après ajustement.

3.2 Calcul de l'intensité du gradient

La deuxième étape consiste à calculer le gradient de l'image filtrée. Les contours correspondent aux zones où le changement de l'intensité lumineuse est important c'est-à-dire, là où la valeur du gradient est la plus grande. Un contour peut pointer dans diverses directions, Canny utilise quatre filtres pour détecter les directions horizontales, verticales et diagonales.

L'estimation du gradient se fait par le filtre de Sobel qui calcul la dérivée première dans la direction horizontale (G_x) et verticale (G_y) grâce à leurs masques respectifs :

$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

On en déduit la norme du gradient en un point :

$$G = \sqrt{G_x^2 + G_y^2}$$

La direction du gradient est donnée par :

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right)$$

L'angle est arrondi à l'un des quatre angles représentant la verticale (0°), l'horizontal (45°) et les deux diagonales (90° et 135°).

3.3 Suppression des valeurs non maximales

Une forte intensité du gradient indique la présence probable d'un contour mais cela n'est pas suffisant. Seuls les maxima locaux du gradient sont conservés dans la direction du gradient, les autres sont supprimés.

A titre d'exemple, si l'angle du gradient arrondi est 0 (donc contour nord-sud), le point sera considéré comme faisant partie du contour à condition que l'amplitude de son gradient est supérieure à celles du pixel précédent et suivant dans la direction verticale.

Il en est de même dans les 3 autres directions (horizontale et les 2 diagonales).

A la fin de cette étape on se retrouve avec une matrice binaire contenant les contours de l'image.

3.4 Seuillage double

Les contours obtenus à l'étape précédente sont dû au bruit restant. Afin de les supprimer, un double seuillage sur l'intensité du gradient est utilisé, un haut et un bas. Le gradient de chaque pixel du contour est comparé à ces seuils. S'il est inférieur au seuil bas, il est supprimé. S'il est supérieur au seuil haut il fait partie du contour. S'il est entre le seuil bas et le seuil haut il est accepté lorsqu'il est connecté à un pixel déjà admis.

Dans notre approche nous avons fixé un seuil bas très petit (0.00001). Nous débutons par une valeur du seuil haut élevée (0.9) tout en restant inférieure à 1.

L'image obtenue perd une grande partie du texte. Nous avons alors abaissé le seuil haut jusqu'à ce que tous les caractères de l'image d'origine soient visibles, c'est à dire 0.4. Une fois le seuil haut fixé on augmente le seuil bas pour supprimer les contours dus au bruit notamment en bas à droite de l'image. Nous parvenons à un bon résultat pour un seuil bas de 0.09, au-delà certaines informations sont perdues comme sur le 7 et le 6 de 1763 ou bien le *f* de *of*.

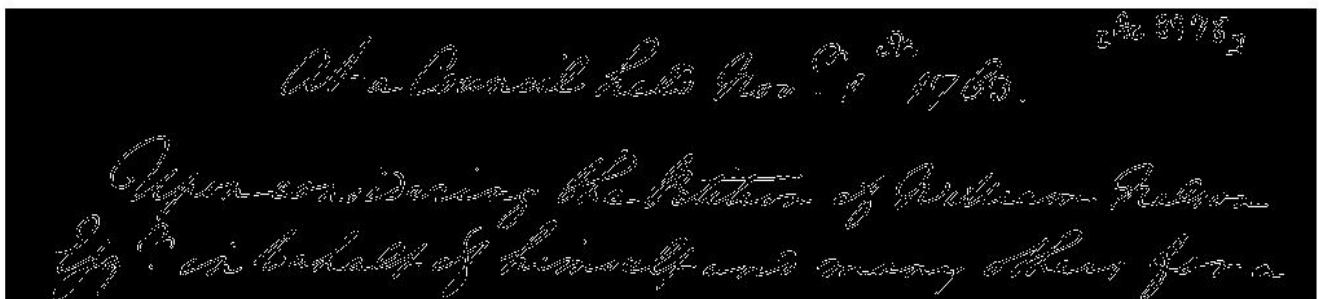


Fig.1 Image du contour

A l'issue de cette étape, on obtient une image binaire avec les pixels appartenant au contour (1) et les autres (0) (fig.1).

3.5 Numérotation des objets

Notre but initial après détection des contours était de « remplir » l'intérieur des contours pour former des objets. Pour cela il fallait inverser les couleurs dans l'image montrant les contours puis numéroté les zones blanches. Ainsi en changeant la couleur du fond en noir tous les objets blancs formeraient le texte. Cependant, le désavantage est que les contours ne sont pas toujours fermés donc les objets qu'ils entourent se confondent avec le fond. Aussi, certaines parties de l'image comme l'intérieur du « o » ou du « 8 » sont détectées comme des objets puisqu'elles sont entourées par un contour alors qu'elles ne le sont pas dans l'image d'origine. Nous avons donc dû abandonner cette idée.

A présent, que nous avons les contours, il faut leur adjoindre les autres pixels formant le texte.

Il faut appliquer un seuillage sur l'image d'origine pour ne conserver que les pixels qui nous intéressent.

Attention, le seuillage ne doit en aucun cas être appliqué sur l'ensemble de l'image au risque de faire apparaître du bruit (point noirs). Pour limiter cela on a numéroté les objets de l'image des contours puis avons noté leurs coordonnées et dimensions. Ensuite on a appliqué le seuil sur les parties de l'image qui leur correspondent. Ainsi on évite de faire le seuillage sur des zones de l'image qui ne nous intéressent pas.



Fig.2 image d'un objet.

3.6 Seuillage

Nous avons sélectionné ce seuil au mieux en nous concentrant sur le « 7 » de « 1763 » car certains pixels qui le forment diffèrent peu en intensité par rapport au fond de l'image. Un seuil égal à 165 nous permet de garder la forme du « 7 ». Pour chacun des objets tous les pixels d'intensité inférieure à 165 feront partie de l'objet, les autres formeront le fond. Ensuite on assigne la valeur 0 aux pixels des objets et 1 à ceux du fond. Le texte est alors affiché en noir et le fond en blanc.

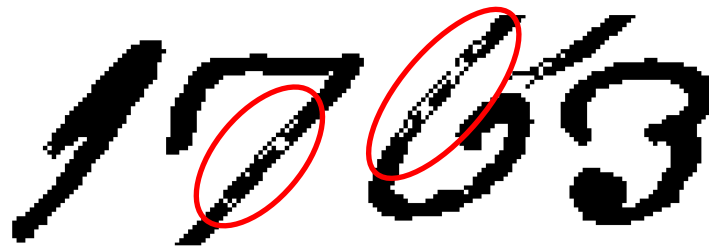


Fig.2 Portion de l'image après seuillage

Après seuillage certaines parties de l'image perdent de l'information, par exemple le « 7 » et le « 6 » (fig.2). Ceci est bien sûr dû à la valeur du seuil, plus il est grand plus on est fidèle à l'image d'origine hormis l'intérieur des lettres comme par exemple le « a » ou le « 8 » qui se retrouvent noirs. Il a donc fallu établir un compromis; on a préféré conserver certains détails comme l'intérieur du « 8 » ou celui du *h* de *himself* par exemple.

3.7 Amélioration de l'image

L'intérieur des objets n'est pas uniforme, on y trouve des pixels blancs éparses qu'on souhaite changer en pixels noirs (entourés sur fig.2).

Notre première méthode a consisté à faire, pour chaque pixel blanc la moyenne de ses 8 voisins. S'il est entouré par 2 pixels blancs ou moins (moyenne égale à 0.25) il devient noir. Ceci marche également pour les pixels isolés, par contre pour des plus gros blocs comme sur le 6 (fig.2) cela n'a pas d'effet.

Pour y remédier on a numéroté ces objets blancs en fixant préalablement le nombre de pixels connectés à 4 ; si un pixel blanc a un de ses voisins (à l'horizontale ou à la verticale) blanc alors ils forment un même objet. Ceci permet de prendre en compte les pixels proches du contour et qui autrement feraient partie du fond blanc.

Par la suite, on a calculé l'air de chaque objet. S'il est inférieur à un certain seuil (égal à 15) on change la couleur de ses pixels en noir. Là aussi il a fallu faire un choix entre remplir le plus de pixels blancs et conserver l'intégrité des caractères cités précédemment.



Fig3. Portion après traitement

3.8 Elimination du bruit persistant

Il persiste encore un peu de bruit caractérisé par des pixels noirs situés près de certains caractères comme le 8 et d'autres un peu partout sur l'image.

Nous avons opté pour le filtre médian qui contrairement à un filtre linéaire comme le filtre moyenneur introduit très peu de flou et préserve les contours. Cependant ce filtre est imparfait et certains pixels noirs qu'on souhaiterait conserver sont supprimés. Par exemple avec un filtre 5*5 le 6 (fig.3) est coupé en deux mais reste lisible, par contre une grande partie du bruit est supprimée.

Son principe est de calculer pour chaque pixel la médiane des valeurs des pixels à l'intérieur de la fenêtre choisie (ici 5*5).



Fig4. Portion avant filtrage et après application du filtre médian

At a Council held Nov^r 1st 1763. ^{the 8978}
Upon considering the Petition of William Nelson
Esq^r in behalf of himself and many others for a

c. 8978,

At a Council held Nov. 1st 1763.

*Upon considering the Petition of William Nelson
Esq. in behalf of himself and many others for a*

Fig5. Images finales sans(en haut) et avec filtrage

➡ Cette méthode nous a permis d'obtenir un résultat assez fidèle à l'image optimale.

Le bruit est quasi inexistant et les endroits les plus difficiles à faire ressortir comme le 7, le 6 ou les f sont bien représentés (fig.5). Cependant cette méthode a ses limites et des différences subsistent avec l'image optimale, notamment l'intérieur du « a » de *many*, du « o » de *for* et du « 8 » qui sont remplis.

Ce qui s'explique par les différents paramètres qu'il faut introduire manuellement et qui résultent d'un compromis entre débruitage et conservation des détails. Mais la partie la plus critique est le seuillage globale appliqué sur tous les objets de l'image. A cause de l'intensité du bruit de fond qui est du même ordre que celle de certaines parties du texte ce seuil ne permet pas d'éviter tout le bruit tout en faisant ressortir tous les détails.

Cela nous a conduits à rechercher une méthode qui pourra calculer ce seuil pour chaque portion de l'image afin de tenir compte des variations en intensité qui diffèrent selon les zones de l'image.

4 Méthode Sauvola

La méthode de seuillage globale, utilise une valeur seuil d'intensité pour séparer les pixels de l'image en deux classes: objet ou fond. Elle n'est pas efficace lorsque le document est de mauvaise qualité (par exemple des erreurs lors du scannage), ce qui est le cas pour les documents anciens.

Chaque objet de l'image ou une partie de celui-ci peut avoir son propre niveau d'intensité pour le séparer du fond. Nous avons été confrontés à cette problématique lors de la 1ère méthode dans laquelle le seuillage a effacé certains détails.

Le seuillage local calcul le seuil localement, pixel par pixel ou région par région.

Pour chaque région le but est donc d'évaluer le seuil optimal pour séparer le texte du fond.

Le plus atteignable, est de le définir manuellement en étudiant l'histogramme de chaque région en couvrant toute l'image. Ceci est évidemment fastidieux surtout pour notre image de dimension 426x2025, si nous la partageons en de petites régions.

Il faut donc calculer le seuil automatiquement, et tenir compte des propriétés de chaque région pour l'optimiser.

Il existe plusieurs méthodes, plus ou moins complexes basées sur le calcul du seuil local : Sauvola, Niblack, Bernsen et Eikvil.

Notre recherche nous a dirigés vers Sauvola, qui est une amélioration de celle de Niblack, afin de réduire sa sensibilité au bruit, car elle donne de bons résultats pour les documents anciens.

4.1 Formule de binarisation.

Le seuil est calculé par la relation suivante :

$$T(x, y) = m(x, y) * \left[1 + k * \left(\frac{s(x, y)}{R} - 1 \right) \right]$$

L'idée est de calculer pour chaque petite portion de l'image en niveaux de gris la moyenne m et l'écart-type de ses pixels. Deux autres paramètres doivent être fournis à l'algorithme: la portée dynamique de l'écart-type R et un autre paramètre fixe k . Ces paramètres ont pour effet d'amplifier la contribution de l'écart-type de façon adaptative. Il n'y a pas de règle pour le choix des valeurs de R et k . L'article de Sauvola sur lequel nous nous sommes appuyés mentionne un $R=128$ et $k=0.5$, cependant pour notre image ils ne donnent pas les meilleurs résultats. Nous les avons donc adaptés pour notre cas.

4.2 Principe de la méthode

- Partage de l'image en fenêtres.

Pour définir les zones, on partage l'image en fenêtres d'égales dimensions.

Sauvola recommande une largeur de $n=10$ à 20 pixels. Une fenêtre plus petite permet de mieux faire ressortir les détails, plus grande elle supprime mieux le bruit. Les essais nous ont permis de trouver un bon compromis avec une fenêtre 13×13 .

Avec des fenêtres de cette dimension il n'est pas possible de couvrir toute l'image ; les fenêtres dépassent des bords droit et bas. On doit donc laisser une bande vide de fenêtres à ces endroits. Nous corrigerons ce problème dans la partie avec traitement.

- Remplissage des fenêtres.

On fait glisser une fenêtre vide $n \times n$ tous les n pixels dans le sens de la longueur et de la largeur pour couvrir l'image tout en évitant de dépasser les bords droit et bas (fig.6). Pour chacune des fenêtres on duplique les $n \times n$ pixels de l'image qui correspondent à l'emplacement de la fenêtre.

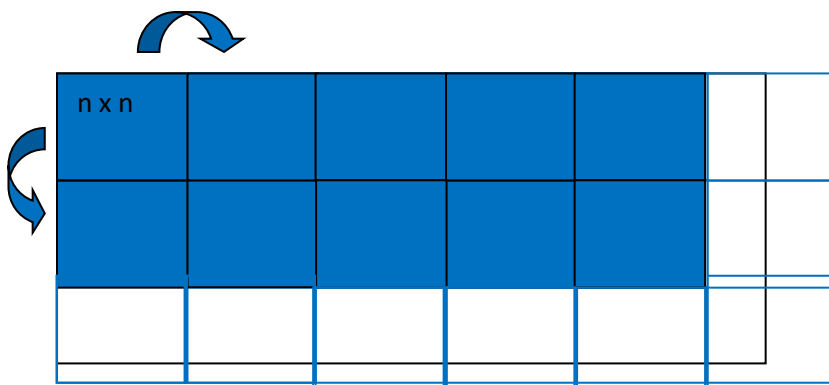


Fig.6 Déplacement de la fenêtre. Fenêtres transparentes excluent.

- Calcul du seuil

Pour chaque fenêtre on calcule la moyenne m de ses pixels et son écart-type s .

Pour la détermination des paramètres R et k , il a fallu appliquer différentes valeurs avant de trouver celles qui donnent le meilleur résultat en terme de rendu final pour notre image. D'après la formule, plus k est grand plus le seuil est petit est par conséquent moins on conserve de détails. Réduire k augmente le seuil et rehausse le bruit. T est moins sensible à la variation de R . On observe plus de détails en le baissant.

Nous avons fixé R sur 128 puis avons diminué k à partir de 0.5 jusqu'à obtention d'une image satisfaisante qui allie peu de bruit et conservation des détails. Par la suite, on a fait varier R . Les meilleurs valeurs pour notre image sont $k=0.04$ et $R=25$.

Une fois les paramètres fixés on calcule le seuil pour chaque fenêtre.

- Application du seuil sur l'image

L'étape finale consiste à comparer chaque seuil local aux pixels de l'image d'origine correspondant à cette zone.

Si l'intensité du pixel est inférieure au seuil, on lui attribue la valeur 0 dans une nouvelle matrice préalablement remplie de 1. Ainsi deux classes seront créées : les objets (0) et le fond (1).

4.3 Application sans prétraitement.

Dans un premier temps nous avons traité notre image sans traitement préalable afin de déterminer son efficacité (fig.7). Ainsi, si on obtient un résultat satisfaisant, son application est plus facile à d'autres images et nous avons moins de paramètres à définir selon l'image.

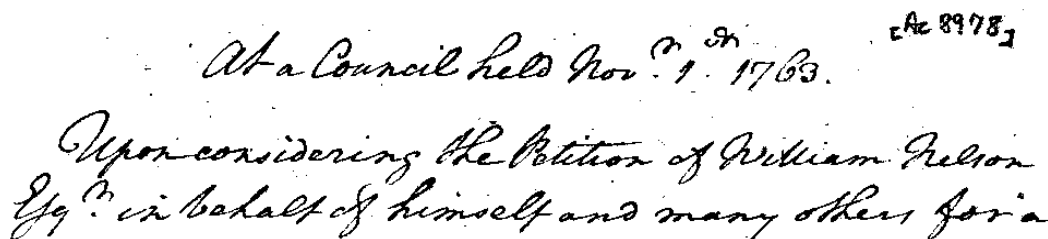


Fig.7 image après application de Sauvola.

L'image obtenue est assez fidèle aux détails cependant le bruit est encore présent (fig.7). Nous appliquons alors en post-traitement un filtre médian 5*5 pour supprimer les pixels noirs de forme « poivre et sel » (fig.8).

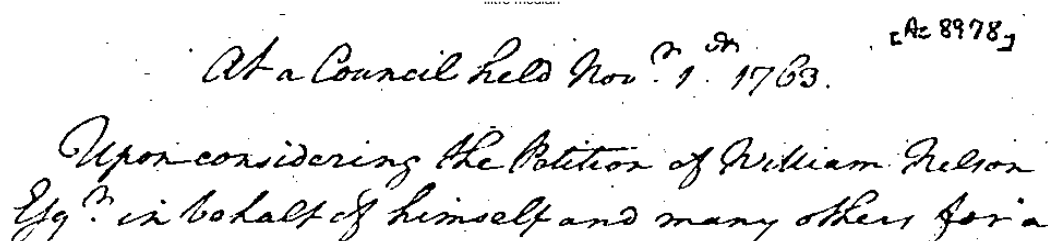


Fig.8 Sauvola suivie d'un filtre médian 5*5.

L'image finale après filtrage (fig.8) n'est pas satisfaisante ; certes on a réduit le bruit mais des tâches persistent encore, celles formant des blocs de pixels plus gros. On utilise alors une autre approche : la reconstruction morphologique. Pour cela on dilate d'abord l'image obtenue avec Sauvola et on inverse ses couleurs. On obtient ainsi le marqueur (fig.9). L'image initiale (fig.8) également inversée servira de masque (fig.10).

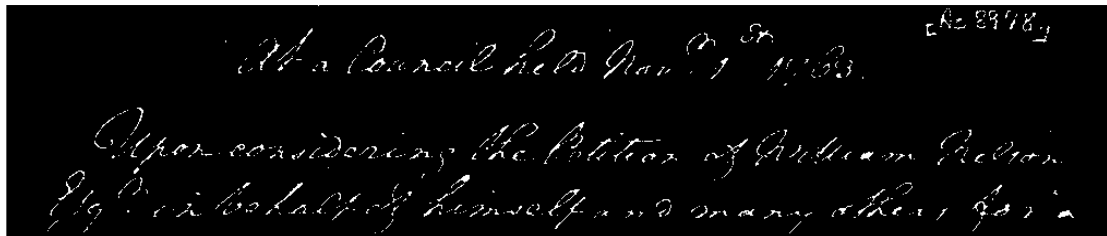


Fig.9 image marqueur.

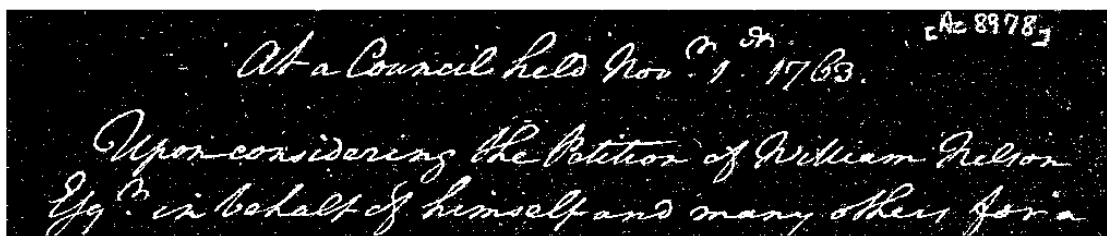


Fig.10 image masque.

La reconstruction morphologique, réalisée sous Matlab avec la commande *imreconstruct* est la dilatation répétée du marqueur jusqu'à ce que le contour de cette image s'inscrive dans l'image du masque. Chaque dilatation, successive doit se trouver au-dessous du masque. Lorsque plus d'une dilatation cesse de changer l'image le traitement s'arrête. La dilatation finale est l'image reconstruite (fig.11).

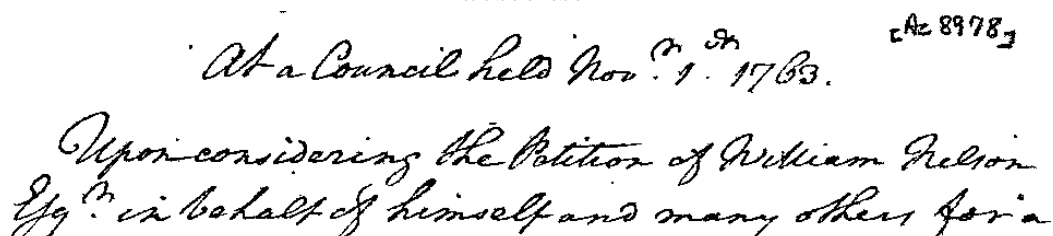


Fig.11 image reconstruite finale.

➡ Cette image est bien meilleure, que celle obtenue avec la deuxième méthode. Les détails apparaissent mieux notamment sur 8978. L'intérieur blanc des lettres et des chiffres est bien conservé. Cependant, ce n'est pas encore parfait en comparaison avec l'image optimale. Il y a encore un peu de bruit et le problème de bord discuté dans la

deuxième partie apparaît en bas de l'image en particulier sur la première lettre. La combinaison de la deuxième méthode et de Sauvola pourrait sensiblement améliorer l'image.

4.4 Application avec prétraitement.

Dans la partie sans prétraitement nous avons utilisé tous les pixels de l'image d'origine. Cela a eu pour conséquence la persistance de certaines tâches difficiles à éliminer sans affecter les détails.

Dans la deuxième méthode, nous avons tracé le contour et conservé ses pixels pour la suite, ce qui a permis d'améliorer la détection de certains détails. Ensuite, nous avons numérotés les objets constituant le contour puis nous les avons traités par un seuillage global. En post-traitement les pixels blancs faisant partie des caractères et qui n'ont pas été considéré à cause du seuillage ont été changé en pixels noirs.

Dans cette partie, il s'agit de remplacer l'étape de seuillage globale de la deuxième méthode par le seuillage local réalisé précédemment afin de combiner leurs qualités. On corrigera également le problème de bord.

Comme dans la deuxième méthode nous appliquons le seuil local objet par objet. Afin d'éviter le problème de bord et traiter chaque pixels des objets on a adapté les dimensions des fenêtres ; quand la fenêtre initialement de taille $n \times n$ (13x13 dans notre cas) dépasse du bord droit ou bas de l'image, on la redimensionne. On définit sa longueur, en diminuant le nombre de colonnes de l'objet par celui de la position du bord droit de la dernière fenêtre. Sa largeur est la différence entre le nombre de lignes de l'objet et la position en nombre de lignes du bord bas de la dernière fenêtre (fig.12).

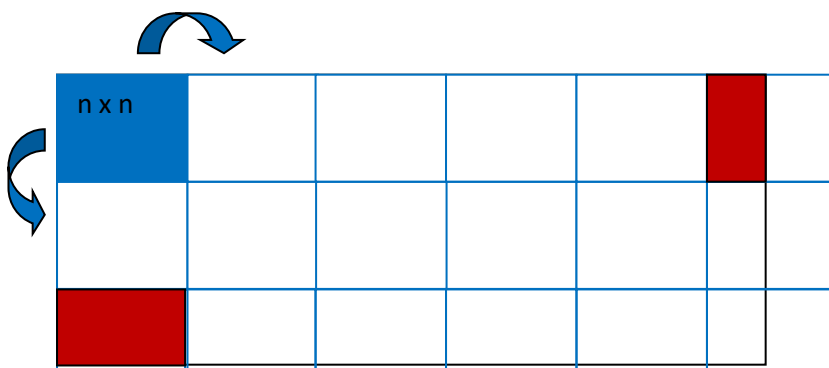


Fig.12 fenêtres de dimensions différentes à $n \times n$ sur les bords.

Cette combinaison de méthode nous a fourni le résultat suivant:

At a Council held Nov^r 1st 1763 [Ac 8978]
 Upon considering the Petition of William Nelson
 Esq^r in behalf of himself and many others for a

Fig.13 résultat final combinant la méthode 2 et sauvola.

Le résultat est très satisfaisant. Les détails ressortent parfaitement et le bruit est quasi inexistant. Il y réside un peu de bruit « poivre et sel » autour de certains caractères. On applique alors un filtre médian 3x3 afin d'éviter de toucher les détails tout en enlevant ces pixels noirs (fig.14).

At a Council held Nov^r 1st 1763 [Ac 8978]
 Upon considering the Petition of William Nelson
 Esq^r in behalf of himself and many others for a

Fig.14 résultat final

At a Council held Nov^r 1st 1763. [Ac 8978]
 Upon considering the Petition of William Nelson
 Esq^r in behalf of himself and many others for a

Fig.15 image optimale donnée

On a comparé notre résultat final avec l'image optimale (fig.16).

Les points rouges représentent les pixels en trop présents. Ils sont localisés uniquement autour des caractères et n'introduisent pas de bruit ce qui rend notre texte plus épais mais n'altère en rien sa lisibilité. Cela peut être dû à l'étape de détection des contours. Par contre il y a peu de points de l'image optimale qui ne soient pas visibles sur notre

image (points bleus).

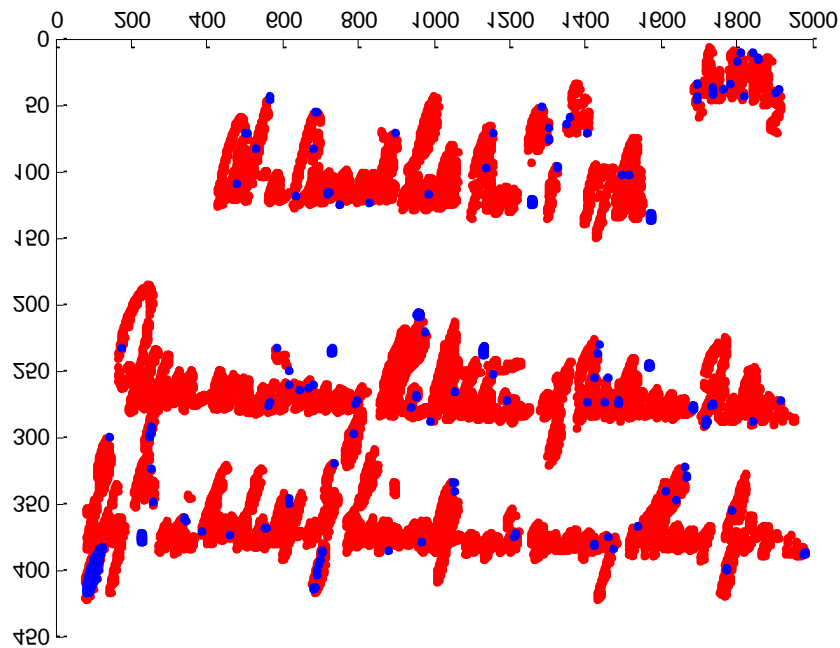


Fig.16 comparaison avec l'image optimale

5 Conclusion

Ce projet nous a permis d'étudier différentes méthodes de binarisation et que leurs application sont très répandue dans le domaine d'indentification de document anciens.

Comme nous l'avons montré dans cette étude il existe de nombreuses méthodes comme par exemple le seuillage globale classique par analyse d'histogramme ou Otsu qui ont montré peu d'efficacité. Cela est dû principalement à l'image traitée qui présente différentes tâches et des objets avec des pixels d'intensité trop proche de celle du fond. Cette méthode est plus appropriée quand les objets et le fond sont distinguables sur l'histogramme. Pour avoir une amélioration significative il a fallu utiliser un seuillage local, en cela la méthode Sauvola de calcul du seuil a est très efficace. Un résultat optimal aurait peut-être nécessité d'explorer la reconstruction par watershed .

6 Bibliographie

- [1] <http://www.mediateam oulu.fi/publications/pdf/24.p>
- [2] <http://www.mathworks.fr/fr/help/images/morphological-reconstruction.html>
- [3] <http://www.mathworks.fr/fr/help/images/examples/marker-controlled-watershed-segmentation.html>
- [4] https://fr.wikipedia.org/wiki/Filtre_de_Canny
- [5] <http://stephanieluu.com/image-convolution/article2/introduction>
- [6] <http://www.mathworks.fr/fr/help/images/ref/edge.html>
- [7] http://www.snv.jussieu.fr/~wboudier/ens/cours_inb2/06_Segmentation.pdf
- [8] <http://www.lesialab.net/IMAGE2009/Documents/Communications/14-Kefali.pdf>
- [9] Gilles Burel. Introduction au traitement d'images. Lavoisier

7 Annexes

Annexe 1

Otsu

```
%% Méthode d'Otsu

close all
clear

%lecture de l'image
O=imread('H01.bmp');

%conversion de l'image en niveaux de gris
Og=rgb2gray(O);

%calcul du seuil par Otsu
level=graythresh(O);

%binarization à partir du seuil
BW1=im2bw(O,level);

%affiche l'image binarisée
figure(1)
imshow(BW1)
title('Otsu seul')
```

Annexe 2

Méthode Canny

```
%Méthode Canny

close all
clear

O=imread('H01.bmp');
Og=rgb2gray(O);

%test de fenêtres gaussiennes
%g=fspecial('gaussian',[5,5],0.83); % [7,7] , 5
%gauss=imfilter(Og,g,'same');
%figure;imshow(gauss)

%détection de contours par canny
%v: vecteur contenant les seuils bas et haut
%sigma=0.7 pour la gaussienne

v=[0.09 0.40001];
can1 = edge(Og,'canny',v,0.7);

figure;imshow(can1);

se = strel('disk', 2);

%numérotation des objets de l'image de contours
L2=bwlabel(can1);

figure;imshow(L2);
title('image des contours')

%définir les positions des objets
stat=regionprops(logical(L2),'BoundingBox');

%nombre d'objets
nL=max(max(L2));

%boucle sur tous les objets
%xy contient la position (xy(1) et xy(2))de l'objet n sur l'image et ses
%dimensions (xy(3) et xy(4))

for n=1:nL
```

```
xy=round(stat(n).BoundingBox);

%boucle sur chaque pixel de l'objet
for i=xy(1):(xy(1)+xy(3))

    for k=xy(2):(xy(2)+xy(4))

        %si le pixel est inférieur au seuil il fait partie de l'objet
        if Og(k,i)<164
            L2(k,i)=n;
        end
    end
end

end

%dimensions de l'image d'origine
[x y]=size(Og);

%si le pixel ne fait pas partie du fond(n=0) il est noir
L3=0*L2+1;
L3(L2>0)=0;

figure; imshow(L3)
title('image après seuillage')

%numérotation des objets(pixels blancs dans les caractères) afin de les
%changés en pixels noirs
Lab=bwlabel(logical(L3),4);

figure;imshow(Lab)
title('labels')

%définir la position et l'air des objets
stat2=regionprops(Lab,'BoundingBox','Area');

nL2=max(max(Lab));

%boucle sur les objets hormis le fond(n2=1)
for n2=2:nL2
    area=stat2(n2).Area;

    %si air<15 l'objet fait partie du texte
    if area<15
        Lab(Lab==n2)=0;
    end
    %area=0;
end

%conversion en image binaire
Lab2=0*Lab;
Lab2(Lab>0)=1;

figure;imshow(Lab2)
title('après remplissage des pixels blancs isolés')

%application d'un filtre médian 4*4 pour enlever le bruit
```

```
med=medfilt2(Lab2,[4,4]);  
figure;imshow(med)  
title('après filtrre médian')
```

Annexe 3

Méthode Canny et Sauvola

```
%Méthode Canny + Sauvola  
  
close all  
clear  
  
I=imread('H01.bmp');  
I2=rgb2gray(I);  
  
figure;imshow(I2)  
  
v=[0.09 0.40001];  
can1 = edge(I2,'canny',v,0.7);  
  
figure;imshow(can1);  
  
L2=bwlabel(can1);  
  
figure(2);imshow(L2);  
  
stat=regionprops(L2,'BoundingBox','Image','Area');  
  
aL=max(max(L2));  
  
%binarisation de l'image des contours  
L3=0*L2+1;  
L3(L2>0)=0;  
  
Im=I2;  
  
%boucle sur chaque objet  
for a=1:aL  
  
    %position de l'objet dans l'image  
    ss=round(stat(a).BoundingBox);  
  
    %parcourt de la l'"objet" à seuiller  
    %on fait glisser une fenêtre 13*13 sur l'image de l'objet  
  
    for i=1:13:ss(4) %boucle sur les lignes  
        for j=1:13:ss(3) %boucle sur les colonnes
```

```

h=i;hi=i+13;
s=j;sj=j+13;

%choix de la taille de la fenêtre et création d'une matrice vide à remplir
%ensuite

%si la fenêtre 13*13 dépasse le bord bas de l'image
%sa largeur=largeur de l'image-position de la dernière fenêtre
%sa longueur=13
if ( hi>ss(4) && sj<=ss(3))
    MI=zeros(ss(4)-h,sj-s);
    hi=ss(4);
end

%si la fenêtre 13*13 dépasse le bord droit de l'image
if ( hi<ss(4) && sj>ss(3))
    MI=zeros(hi-h,ss(3)-s);
    sj=ss(3);
end

%si la fenêtre est dans le coins en bas à droite
if ( hi>ss(4) && sj>ss(3))
    MI=zeros(ss(4)-h,ss(3)-s);
    hi=ss(4);
    sj=ss(3);
end

%si la fenêtre est à l'intérieur de l'image
if ( hi<=ss(4) && sj<=ss(3))
    MI=zeros(hi-h,sj-s);
end

%copie de la portion choisie de l'image dans cette fenêtre
o=1;
p=1;

for m=h:hi
    for n=s:sj

        %(-1) pour bien prendre en compte tous les pixels de
l'image
        %de l'objet
        MI(o,p)=Im(ss(2)+m-1,ss(1)+n-1);

        p=p+1;

    end
    p=1;
    o=o+1;
end

%valeurs de MI sur une colonne
M=MI(:);

%moyenne des pixels de la fenêtre
moyen = mean(M);
%son ecart-type

```



```
deviation=std(M);

%%calcul du seuil pour la portion 15*15

%paramètres pour le calcul du seuil
k=0.04;
R=25;

seuil= moyen*(1 + k*( ( deviation./R )-1));

%application du seuil sur l'image
%Im: image d'origine
%L3: image finale
u=1;
v=1;
    for u=h:hi
        for v=s:sj

            %si le pixel est inférieur ou égale
            %au seuil il fait partie du texte(pixel noir) dans l'image
finale
            if Im(ss(2)+u-1,ss(1)+v-1)<=seuil
                L3(ss(2)+u-1,ss(1)+v-1)=0;

            end
        end
    end
end

end

figure,imshow(L3);
title('image après seuillage sauvola')
mf=L3;

%numérotation des objets en vu de les changer en pixels noirs après
%détermination de leurs aires

Lab=bwlabel(logical(L3),4);
% figure;imshow(Lab)
% title('labels')

stat2=regionprops(Lab,'BoundingBox','Image','Area');

nL2=max(max(Lab));

for n2=2:nL2
    area=stat2(n2).Area;

    if area<7
        Lab(Lab==n2)=0;
    end
    area=0;
end
```

```
figure;imshow(Lab)
title('après remplissage des pixels blancs isolés')

%élément structurant
se = strel('disk', 3);

%dilatation de l'image
imd = imdilate(Lab, se);
figure, imshow(imd)

%reconstruction de l'image(mask) par sa dilatée(marker)
recons = imreconstruct(imcomplement(imd), imcomplement(Lab));

%inversion des couleurs pour avoir le texte en noir
recons = imcomplement(recons);

figure, imshow(recons), title('image après reconstruction')

%application d'un filtre médian pour enlever le bruit autour des caractères
med=medfilt2(recons,[3,3]);
figure;imshow(med)
title('reconstruction + filtre médian')

%différences entre l'image finale et l'image optimale donnée
figure;
orig=imread('H01_GT.png');
diff=med-orig;

hold on
[x,y] = find(diff==-1);
plot(x,y,'r.')
[x,y] = find(diff==1);
plot(x,y,'b.')
hold off
```

