

Table des matières

1 Généralités sur la segmentation de texture	2
1.1 introduction	2
1.2 Définition de la texture	2
1.3 Segmentation d'images	3
1.4 Les approches analytiques	4
1.4.1 Textures structurelles	5
1.4.2 Textures aléatoires	5
1.4.3 Textures directionnelles	6
1.5 Motif binaire local (LBP-Local Binary Pattern)	6
1.5.1 LBP et le niveau de gris	9
1.5.2 Variantes de la LBP	10
1.6 Conclusion	10
2 Conception	12
2.1 Introduction	12
2.2 Vue global de notre approche	12
2.3 Méthode de découpage	12
2.3.1 Définition	12
2.3.2 Algorithme de découpage dynamique	13
2.4 Généralisation de la méthode de recherches dynamique	14
2.4.1 Algorithme de généralisation	14
2.4.2 Interprétation de l'algorithme	15
2.4.3 Variantes	18
2.5 Création des images de test	21
2.6 Conclusion	22
3 Application et tests	23
3.1 Introduction	23
3.2 Outils utilisés	23
3.3 Presentation et test de l'application :	23
3.4 Tests et interprétation des résultats :	27
3.4.1 Tests supplémentaires	31
3.5 Conclusion	31

Introduction générale

Il est bien simple et évident pour nous, humains, de reconnaître ce que nous voyons instantanément. Cependant, ce n'est pas aussi simple pour un ordinateur. Étant donné la puissance et la compléxité des traitements effectués par notre cerveau, il nous est très difficile de les analyser, et d'autant plus à les reproduire.

Si on prend en exemple la reconnaissance et la segmentation des textures qui sont aussi une partie du domaine du traitement d'images, nous nous retrouvons dans le paradoxe de l'étude d'une entité non défini mais parfaitement reconnaissable. Le fait de reconnaître un ciel bleu et le distinguer d'une rivière ou des roches tout au tour est élémentaire pour le cerveau humain et utilisé instinctivement par ce dernier. On peut aussi très facilement distinguer une table en plastique d'une table en bois. Cependant, la simple définition du terme «*texture*» reste ambiguë.

Comment pouvons nous donc traiter une entité indéfinie ?

La réponse se trouve dans les études statistiques sur les différentes relations qui existent entre les pixels qui forment une image. Les chercheurs ont pu trouver différentes formules et caractéristiques afin de définir et segmenter les textures. La plupart du temps, l'étude s'intéresse surtout à la détection de motifs réguliers formant la texture, et cela en comparant chaque pixel avec la disposition de son entourage.

La plupart des travaux fait dans ce domaine jusque là, visent à reconnaître une texture dans une image en ayant initialement une partie de cette première comme référence. Ceci est employé dans la recherche de motif déjà connu.

Notre travail consiste à proposer une généralisation de ces méthodes afin de pouvoir l'implémenter dans des systèmes autonomes. Ces derniers doivent être capables d'extraire automatiquement les informations d'une image digitale. En d'autres termes, ils doivent pouvoir segmenter les différentes textures qu'on leur présente.

Notre mémoire est organisé en trois chapitre. Nous présenterons dans le premier chapitre des généralités, des définitions et des notions de bases sur les méthodes utilisées dans notre travail. Dans le second chapitre nous introduirons en détails le principe de l'approche utilisée pour la segmentation des textures. Le troisième sera consacré à la présentation de notre application ainsi que les testes réalisés sur des images générées aléatoirement.

Chapitre 1

Généralités sur la segmentation de texture

1.1 introduction

Dans ce chapitre, nous examinerons les différentes définitions de texture proposés par plusieurs chercheurs du domaine. Puis nous verrons les notions de bases du traitement d'images et un aperçu de ses domaines d'utilisation . Nous découvrirons enfin, plus en détails, la méthodes que nous avons utilisés pour caractériser une textures.

1.2 Définition de la texture

Certaines définitions de la texture ont été sélectionnées et regroupées dans un catalogue par Coggins [COG 82]. On peut en citer quelques exemples :

- "Nous pouvons considérer une texture comme ce qui constitue une région macroscopique. Sa structure est simplement attribuable aux motifs répétitifs dans lesquels des éléments ou des primitives sont disposées selon une règle de placement." [Tam et.al 78]
- "La texture est défini pour nos fins comme un attribut d'un domaine n'ayant aucune composante qui apparaissent dénombrable. Les relations de phase entre les composants ne sont donc pas apparentes. Le terrain ne devrait pas contenir un gradient évident. Le but de cette définition est d'attirer l'attention de l'observateur sur les propriétés globales de l'affichage -. C'est à dire sa "rudesse" globale "bosselage", ou "finesse". Physiquement, les modèles innombrables (apériodiques) sont générés par stochastique par opposition de processus déterministes. A la perception, cependant, l'ensemble des motifs sans composants énumérables évidentes comprendra de nombreux (et même périodique) textures déterministes." [Ric et.al 74]
- "La texture est une notion apparemment paradoxale. D'une part, elle est couramment utilisée dans le traitement de l'information visuelle, surtout pour des raisons pratiques de classification. D'autre part, personne n'a réussi à produire une définition communément acceptée de la texture. La résolution de ce paradoxe, dépendra d'un modèle plus riche, plus développé pour le début de traitement de l'information visuelle, un aspect central de ce qui sera un systèmes représentatif à différents niveaux d'abstraction. Ces niveaux vont probablement inclure _____ intensités actuels au fond et progresseront à travers des pointes et des descripteurs d'orientation vers la surface, et peut-être des descripteurs volumétriques. Compte tenu de ces structures multi-niveaux, il semble clair qu'ils doivent être inclus dans la définition, et dans le calcul des descripteurs de la texture." [Zuc et.al 81]
- "La notion de texture semble dépendre de trois ingrédients : (1) un certains « ordre » local répété sur une région qui est grande par rapport à la taille de l'ordre, (2) l'ordre consiste en l'arrangement non aléatoire de pièces élémentaires et (3) les parties sont des entités plus ou moins uniformes ayant approximativement les mêmes dimensions partout dans la région texturée." [Haw 69]

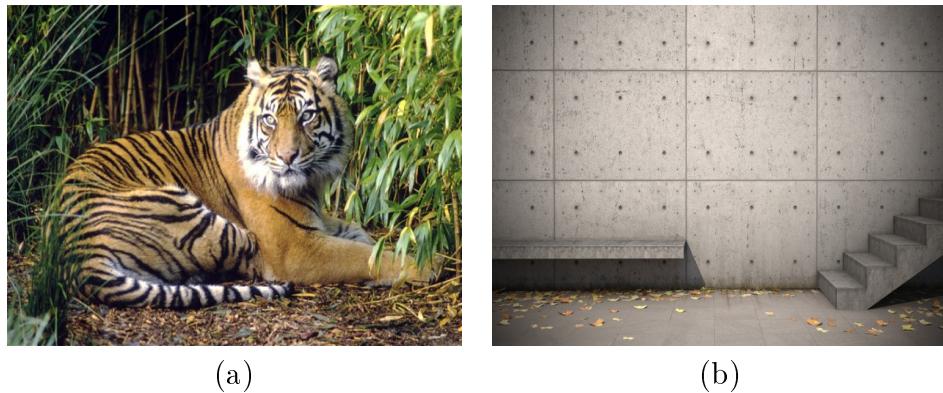


FIGURE 1.1 – Différentes textures

Quelques paramètres essentiels sont à prendre en compte dans l'étude des textures. Si on prend par exemple la photo d'une forêt, on pourra distinguer la terre comme texture, l'ensemble des feuilles d'un arbre comme une autre, mais encore la peau d'un tigre [Figure 1.1](a).

Si on prend par contre une prise aérienne de cette même forêt, on pourra considérer le tout comme une seule texture qui se différenciera de la texture formée par les bâtiments d'un village par exemple. Aussi, le voisinage d'une texture joue un grand rôle dans l'interprétation de cette dernière. Une couche de béton par exemple seule peut avoir plusieurs interprétations, elle peut être une partie d'un mur, un banc, le sol ou bien des marches d'escalier [Figure 1.1](b).

Notons aussi que certaines structures ont une dynamique importante au fil du temps qui fait varier leur apparence. Par exemple un ciel bleu avec quelques nuages (le vent fera constamment varier la forme des nuages).

1.3 Segmentation d'images

Dans le traitement d'image, une des plus grandes préoccupations est la segmentation d'images. C'est, comme son nom l'indique, le processus de partitionnement d'image digitale en différentes parties et ce, pour simplifier la « compréhension » de cette dernière par la machine, et la rendre plus facilement analysable.

Le principe est de reconnaître des objets ou des frontières. Le but étant de donner une signification aux pixels de l'image. Elle est généralement utilisée dans :

- La vision artificielle [Figure 1.2](a)
- L'imagerie médicale [Figure 1.2](b)
- La reconnaissance faciale [Figure 1.2](c)
- La reconnaissance d'empreintes digitales [Figure 1.2](d)
- La détection de piétons [Figure 1.2](e)

- La vidéo surveillance [Figure 1.2](f)
- Localiser des objets dans des images satellitaires [Figure 1.2](g)

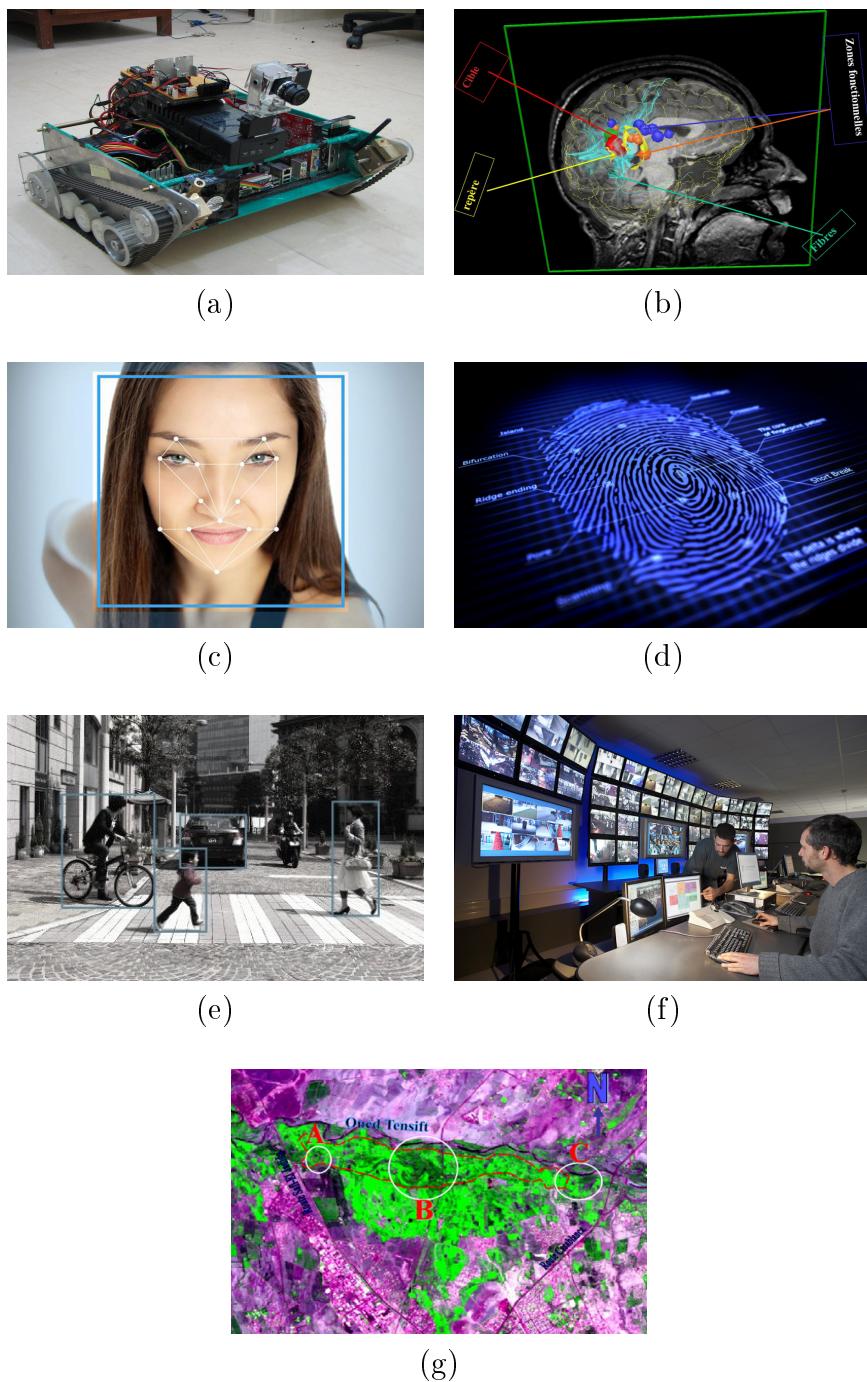


FIGURE 1.2 – La segmentation d’images

1.4 Les approches analytiques

On retrouve principalement quatre (4) approches analytiques de texture qui sont [MAT et STR 98] :

- *Approche structurelle (ou fréquentielle)* : Ces méthodes supposent que les textures sont formées d'éléments structurants de base. L'idée générale de ces méthodes est une recherche et une description des éléments structurants suivie d'une étude de la répartition spatiale de ces derniers [MAJ 09].
- *Approche modèle* : Elle se repose sur l'utilisation de fractals et de modèles stochastique.
- *Approche transformée (méthodes de traitement du signal)* : Des recherches de psychophysiologie ont démontré que le cerveau humain fait une analyse de fréquence des images.
- *Approche statistique* : Ici, la texture est reconnue en utilisant des caractéristiques telles que les "Motifs Binaires Locaux" en se basant sur la distribution des niveaux de gris dans l'image, et c'est sur quoi sera basée notre approche.

L'utilisation de ces approches diffère selon le type de texture. Ces dernières se regroupent en trois différentes catégories : structurelles, aléatoires et directionnelles.

1.4.1 Textures structurelles

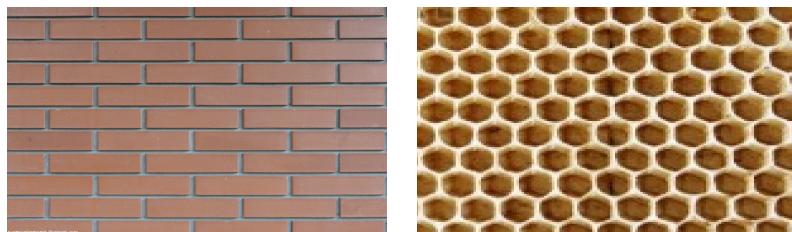


FIGURE 1.3 – Textures structurelles (périodique)

La figure 1.3 présente un premier type de textures dites structurelles. On les appelle ainsi car on peut les considérer comme étant la répartition spatiale de motifs élémentaires de base dans différentes directions de l'espace tout en respectant une certaine règle de placement. En effet, on s'aperçoit que la première représente un mur de brique, elle est composée d'un ensemble d'éléments de base (les briques) disposés relativement régulièrement de manière horizontale. La deuxième texture est aussi composée de motifs de base alvéolés agencés d'une manière particulière les uns à côté des autres [MAJ 09]. Cette catégorie de textures a engendré des méthodes d'analyse dites structurelles.

1.4.2 Textures aléatoires

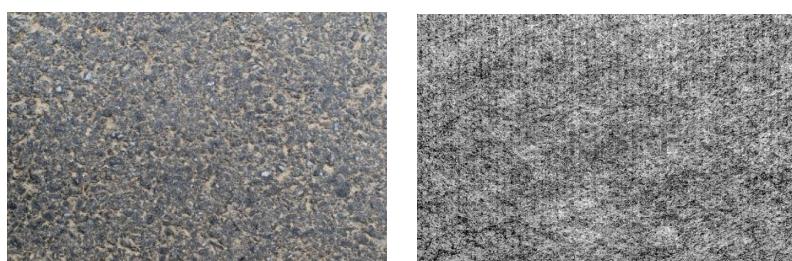


FIGURE 1.4 – Textures aléatoires

Le deuxième type de textures est illustré dans la figure 1.4. Contrairement au premier type de texture, celles-ci ont un aspect anarchique tout en restant globalement homogènes. Ces textures ne peuvent pas être décomposé en des motifs de base se répétant spatialement. On les appelle des textures aléatoires. Cette catégorie a fourni d'autres travaux de recherches plutôt fondés sur des méthodes d'analyse statistique [MAJ 09].

1.4.3 Textures directionnelles

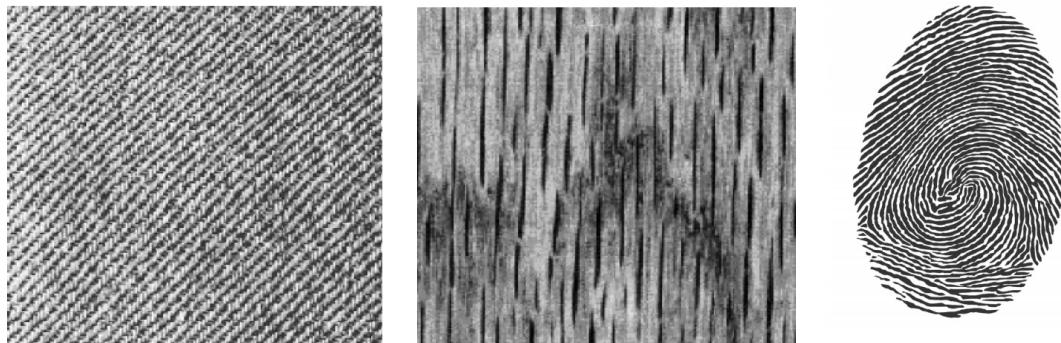


FIGURE 1.5 – Textures directionnelles

Les textures directionnelles [Figure 1.5] ne sont pas forcément aléatoires et ne présentent pas d'éléments structurants de base. On peut les décrire de manière générale comme étant des textures présentant des structures orientées selon une direction privilégiée. Les textures directionnelles se caractérisent soit par leur orientation globale, soit par un champ d'orientations locales. par exemple la texture de gauche de la figure 1.5 laisse apparaître des lignes obliques, et celle au milieu possède des lignes verticales. Dans les textures directionnelles réelles, le champ d'orientation n'est en effet pas toujours uniforme comme la texture de droite de la figure 1.5.

Les différentes catégories de textures qui ont été présenté nous montrent qu'il est difficile de donner une définition "universelle" de la texture. Cependant, on peut décrire le type de la texture selon la distribution et les relations entre les pixels qui la forment.

Nous allons nous approfondir sur l'approche statistique étant donné qu'elle sera celle qu'on implémentera dans notre étude. Plus précisément, nous étudierons la méthode suivante dite **motif binaire local**.

1.5 Motif binaire local (LBP-Local Binary Pattern)

Cette caractéristique a été premièrement évoqué par Harwood [Har et.al 93] et a été utilisé après par Ojala et Pietikinen [Oja et.al 96, Oja and Pie 99]. La caractéristique LBP présente un énorme avantage grâce à sa simplicité de calcul et sa robustesse dans la détection de textures. Elle permet de décrire la texture suivant la configuration

locale des pixels.

Le principe de calcul de la LBP basique (*Basic Local Binary Pattern*) réalisé par Ojala [Oja et.al 02] est de comparer un pixel se trouvant au centre avec ces 8 voisins comme le montre la figure suivante :

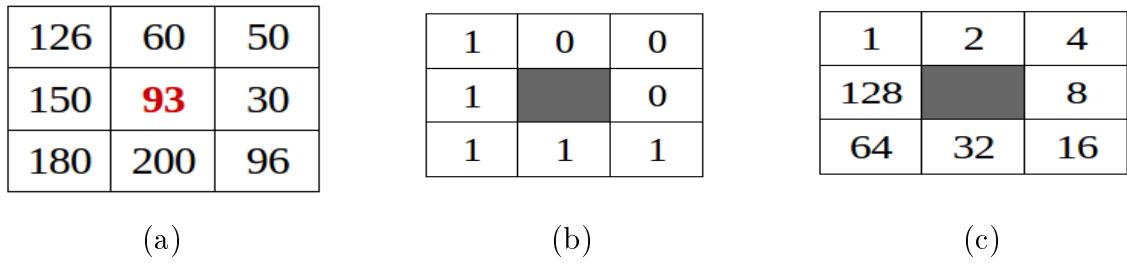


FIGURE 1.6 – Calcul du motif binaire

La matrice I de la figure 1.6 (a) représente le niveau de gris d'un pixel au centre et de ses huit voisins. La matrice M du *motif* qui est représenté dans la figure 1.6 (b) est calculé comme suit :

$$M(i, j) = \begin{cases} 1 & \text{si } I(i, j) \geq g_c \\ 0 & \text{sinon} \end{cases}$$

Où $I(i, j)$ est la valeur du niveau de gris des huit voisins, et g_c qui est le niveau de gris du pixel central (en rouge), le résultat est alors affecté sous format binaire dans les $M(i, j)$.

En lisant les valeurs de $M(i, j)$ en commençant de la case qui se trouve en haut à gauche et en suivant l'ordre de rotation de l'aiguille d'une montre, on obtient le motif binaire m suivant :

$$m = 10001111$$

Ce résultat sera ensuite converti en décimal et cela en multipliant chaque chiffre par les valeurs $P(i, j)$ case par case de la matrice de Poids de la [Figure 1.6](c), ce qui nous donne la valeur LBP du niveau de gris du pixel central qui est :

$$LBP = 128 + 8 + 4 + 2 + 1 = 143$$

La figure suivante montre la transformation LBP d'une image :

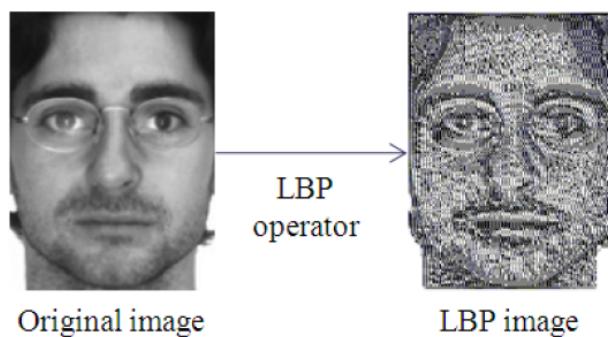
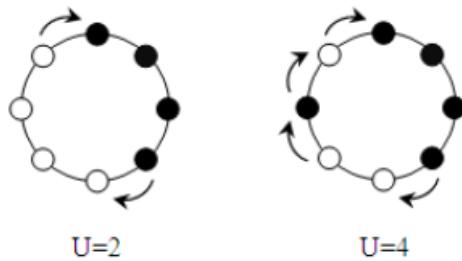
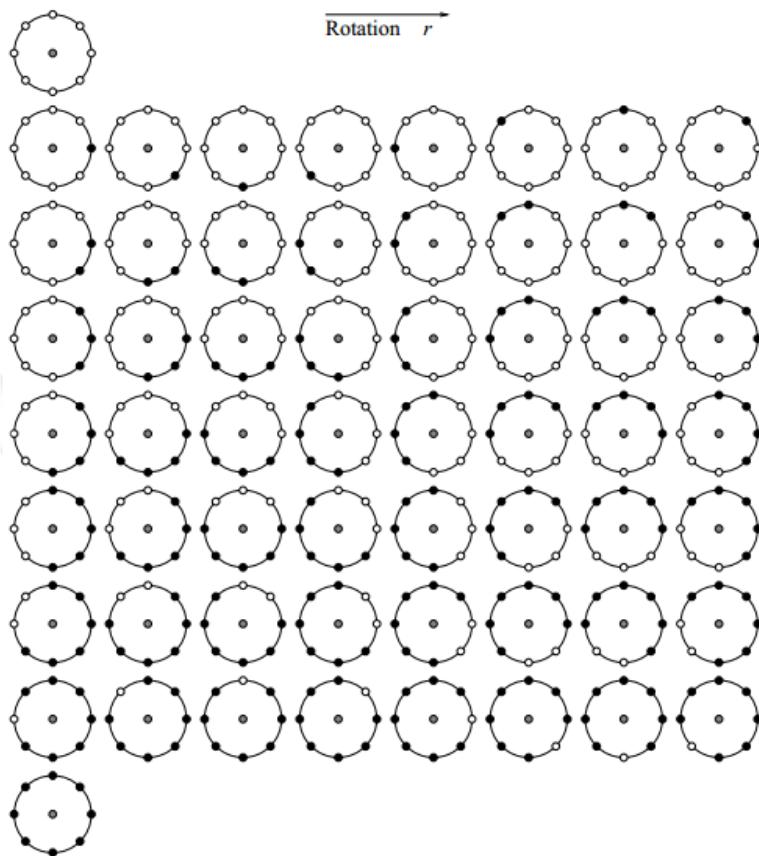


FIGURE 1.7 – Transformation LBP de l'image

Ojala [Oja et.al 02] a aussi définie une valeur " U " qu'on appellera valeur de «*transition*» qui sera égale au nombre de transitions de 0 vers 1 et de 1 vers 0 qu'on trouve dans la valeur du «*motif binaire*» comme suit :


 FIGURE 1.8 – Extraction du nombre U à partir du motif

En réalité, tous les «*motifs*» (256) ne sont pas importants pour décrire une texture. D'après T. Menp [Men et.al 00], les motifs qui ont un nombre de transition $U = 2$ et $U = 0$, et qui sont appelés «*uniform patterns*», représentent une information relative à des motifs réguliers d'une image, autrement dit une texture. Et donc, certaines zones d'intérêt tels des coins ou des bords peuvent être détectées par ce descripteur. Ces motifs sont aussi plus résistants aux transformations géométriques telle que la rotation. L'utilisation des 58 «*uniform patterns*» suivants a été proposé :


 FIGURE 1.9 – Les «*uniform patterns*».

Pour étudier l'uniformité d'une texture, on construit un histogramme qui est composé de 59 valeurs qui sont les 58 «*uniform patterns*» définis plus haut, plus une dernière valeur regroupant tous les «*non uniform patterns*».

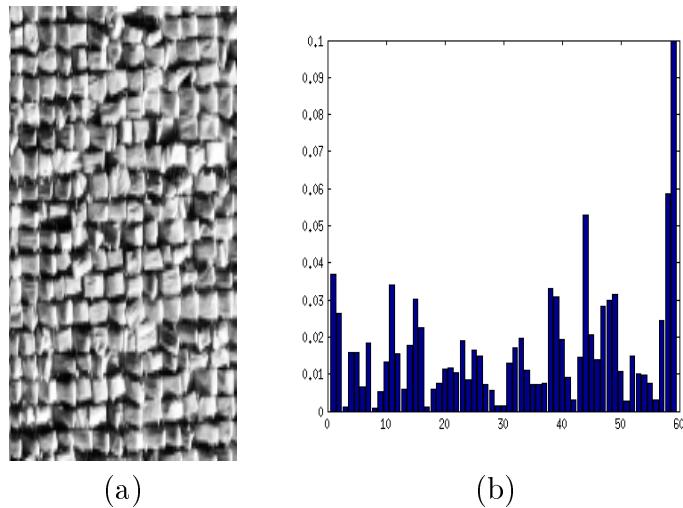


FIGURE 1.10 – Une texture et son histogramme LBP

On compare deux histogrammes LBP de deux textures selon un pourcentage de ressemblance qu'on appellera «*seuil*» ou encore «*Threshold*». On pourra ainsi dire si ces deux textures représentent la même texture ou bien non grâce à cette valeur.

La caractéristique LBP qui utilisent ce genre d'histogramme est noté en tant que LBP^{u^2} . Cette notation est du au fait qu'elle se base sur les «*uniform patterns*» qui ont un $U \leq 2$, par la suite, on notera la LBP^{u^2} en tant que LBP.

1.5.1 LBP et le niveau de gris

L'un des avantages de la LBP est sa robustesse vis à vis du changements monotones du niveau de gris. Ainsi, si on augmente ou diminue le niveau de gris de chaque pixel d'une image d'une même valeur x , cela n'affectera pas l'histogramme LBP créée. Cet avantage réside dans la manière de calcul du nouveau niveau de gris LBP d'un pixel.

Soit un pixel P qui va avoir une nouvelle valeur modifiée par x , il sera comparé avec ses voisins V_i (sachant que ses voisins sont aussi affecté par la valeur x) et on sait que :

$$\text{si } P > V_i \text{ alors } P + x > V_i + x$$

Et comme la comparaison n'a pas changé, alors la nouvelle valeur de niveau de gris du pixel centrale restera toujours la même, comme le montre la figure 1.10.

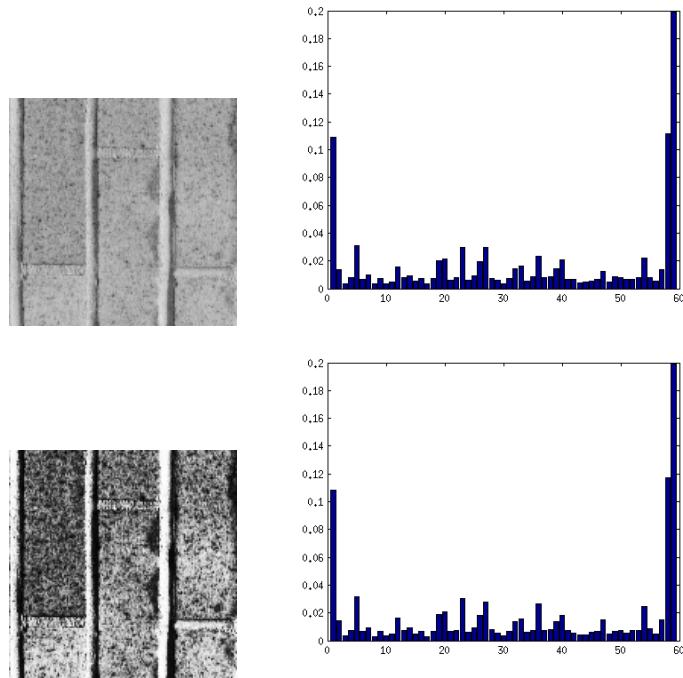


FIGURE 1.11 – Histogramme d'une texture subissant un changement monotone du niveau de gris

1.5.2 Variantes de la LBP

Il existe d'autres variantes de la LBP qui divergent selon le nombre P de voisins dans un rayon R , comme le montre la figure 1.11.

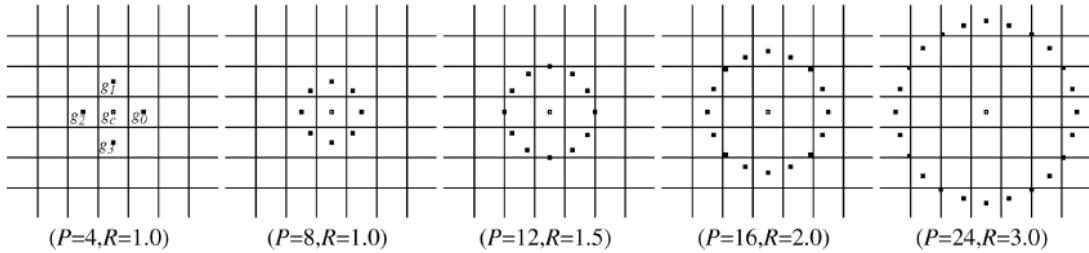


FIGURE 1.12 – Variantes de la LBP

Il a été noté que dans la LBP Basique ($P = 8$ et $R = 1$), le taux des «uniform patterns» est presque de 90%, alors que pour la LBP(16,2) ($P = 16$ et $R = 2$) ce taux ne dépassait pas les 70%. [Oja et.al 02] De ce fait, on se basera sur la LBP Basique qui offre une meilleure détection des textures.

1.6 Conclusion

Dans ce chapitre nous avons décris la texture et ses différents types. Nous constatons que malgré le fait qu'il n'existe à priori toujours pas de définition universelle de la texture, les méthodes de description structurelles restent largement les plus utilisés.

Nous avons présenté le **Motif binaire local** qui est une caractéristique très populaire

et récemment employée pour détecter des personnes et dans la reconnaissance faciale. Dans le chapitre suivant, nous allons parler de son utilité dans notre travail.

Chapitre 2

Conception

2.1 Introduction

Nous présentons dans ce chapitre notre approche qui se base sur la méthode de segmentation dynamique. Nous décrirons cette méthode ou, plus précisément, nous nous intéresserons à son architecture de découpage que nous exploiterons dans notre algorithme. Nous terminons avec des essais de quelques variantes.

2.2 Vue global de notre approche

La méthode LBP offre effectivement un excellent outil pour caractériser une texture. Cependant, son rôle s'arrête là. Pour pouvoir détecter efficacement les textures, il nous faudra appliquer un algorithme de segmentation. Nous avons porté notre choix sur l'algorithme de segmentation dynamique [HAM et.al 13] qui forme une étape primordiale de notre travail de détection. Il se nomme de la sorte car il se base sur des tailles de segments variées (nous expliquerons dans ce qui suit notre choix).

Cependant cette méthode ne permet que de chercher une unique texture qui est connue déjà à l'avance. Notre méthode présente alors une généralisation qui permet de segmenter l'image et détecter toutes ses textures, et ce, en ayant aucune information au préalable, à part l'image elle-même.

Notre méthode a pour but de détecter chaque texture dans les différents endroits où elle est présente dans une image, sans affecter les surfaces qu'occupent les autres textures. On représente cette détection en coloriant avec une couleur unique les surfaces qui contiennent une même texture.

Nous fixons une taille d'échantillon " t ". Nous prendrons comme premier lieu un carré de cette taille au point (0px,0px) qui contient une première texture, et nous chercherons des segments similaires. Nous colorions tous les segments de la même couleur " C ". Ensuite, nous feront de même pour le prochain carré de taille " t " non colorié.

Plusieurs résultats très satisfaisants ont pu être obtenus avec cette approche, et différentes variantes de notre méthode ont été appliquées.

2.3 Méthode de découpage

2.3.1 Définition

Une méthode de découpage est une opération visant à rassembler des pixels entre eux suivant des critères pré-définis. Dans notre cas, les critères de similitudes seront basés sur la ressemblance des textures, définie elle par les paramètres de la méthode LBP (I.4.B).

2.3.2 Algorithme de découpage dynamique

L'algorithme sur lequel nous allons nous baser est décrit dans l'algorithme 1 [HAM et.al 13]. Nous nous basons sur celui-ci pour notre méthode de généralisation. Nous supprimons, le test de ressemblance à la texture de référence [Algorithme 1 - 4.c] pour sauvegarder tous les carrés générés.

Nous supprimons cependant la phase de coloriage dans le test (6) [Algorithme 1 - 6] pour la remplacer par un *return* qui nous retournera le tableau de tout les carrées sauvegardés.

Le processus de division est expliqué dans la figure suivante. [Figure 2.1]

Nous avons choisi d'appliquer cet algorithme dans le but de proposer une solution de généralisation pour la recherche et la détection de plusieurs textures différentes qui sont présentes dans une image, car son aspect dynamique lui offre une bonne robustesse dont on peut citer :

- Des carrés de tailles différentes peuvent être extraits et analysés, grâce à leurs tailles dynamiques. L'ensemble comparé avec la fenêtre principale serait donc très riche et varié
- Différentes formes hormis les carrés et différents points de convergence peuvent être considérés. Par conséquent, différentes configurations peuvent être étudiées.
- L'indépendance entre l'architecture proposée et la méthode d'extraction des caractéristiques est le principal avantage de notre approche. Ce qui permet à cette méthode de s'adapter à différents cas d'utilisations.

```
Function Square dynamic decomposition system(img : image, textureRef : image) : void
    1 Choose the converging pointa.
    2 Consider the main window
    3 Generate as many windows as possible(with the same size of MW)
    4 For each window do
        a Apply the feature extraction method (LBP method);
        b Calculate the feature vector (LBP histogram);
        c Calculate the similarity Sim between the window's feature with the sought
            texture feature
        d If (Sim is above the threshold) then
            | Save the window's position and its feature vector V
        end If
    end For
    5 Reduce the size of MW by the distance d
    6 If ( the size of MW is below the minimum size ) then
        | Color the saved windows and terminate the process
    else
        | Goto step 3
    end If
End
```

Algorithm 1: Algorithme de découpage dynamique

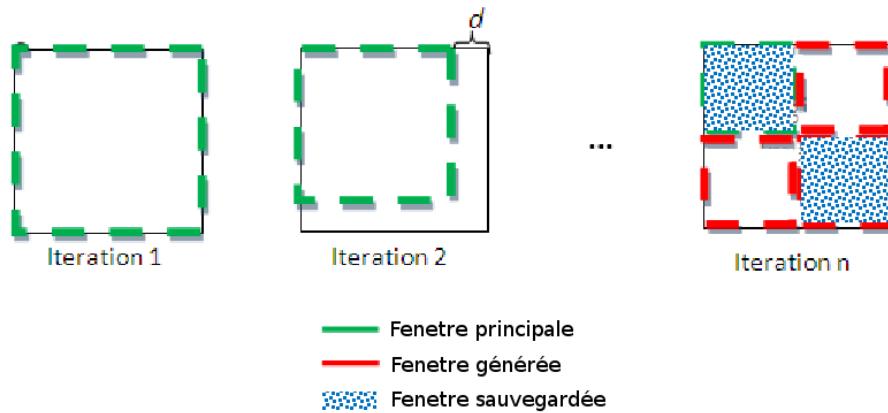


FIGURE 2.1 – Processus de division.

2.4 Généralisation de la méthode de recherches dynamique

Nous allons maintenant aborder tous les détails de notre approche de généralisation de la méthode cité ci-dessus.

L'objectif étant de développer une méthode automatique de détection de toutes les textures présentes dans une image donnée, tout en minimisant au plus l'intervention de l'utilisateur. Chaque texture sera coloriée d'une couleur aléatoire et unique.

2.4.1 Algorithme de généralisation

```

Function Generalization Algorithm(img : image) : void
    1 Constructing a blanc image with the same dimensions as img
    2 Applying the Square Dynamic Decomposition System without colouring
    3 Save the result in a file
    4 Set up a minimal size t and a threshold.
    5 While ( White window of dimension t exists) do
        a Brows the image from right to left, and from top to bottom, and extract the
        first white window Ref with the dimension t
        b Calculate the LBP histogram of Ref
        c For each square do
            calculate the resemblance Resmb of the LBP histogram of the square
            with Ref's histogram
            If (Resmb > threshold) then
                | Colour the correspondent surface in the blanc image.
            end If
        end For
    done
End

```

Algorithm 2: Generalization Algorithm

2.4.2 Interprétation de l'algorithme

Dans un premier lieu, nous nous contenterons d'appliquer l'algorithme sur des images simples.

Une image blanche IB de même taille que l'image à découper est générée [Figure 2.2]. C'est cette première qui sera coloriée progressivement et rendu comme résultat final de l'opération.



FIGURE 2.2 – Initialisation.

L'algorithme de découpage présenté précédemment est appliqué. Ce dernier nous retournera un ensemble de carrés que nous allons sauvegarder dans un fichier. La sauvegarde contiendra les coordonnées de l'extrémité initiale du carré (en haut à gauche) (x_0, y_0) , sa taille t et la valeur de son histogramme LBP h . Le résultats sera un tableau avec pour chaque carrée un élément : $elem = [x_0, y_0, t, h]$

Ce qui nous permettra de ne plus avoir à refaire cette étape les prochaines fois qu'on traitera la même image.

Nous fixons une taille t_r pour les fenêtres de références que nous allons choisir à partir de l'image. Dans notre algorithme, nous l'avons fixé à $30px$.

Nous allons chercher une fenêtre vide de taille t_r . Cette dernière est représentée par un carré non encore colorié de la même taille et à la même position dans l'image blanche créée au début. Nous parcourons pour cela l' IB de gauche à droite, et de haut en bas.

Si la fenêtre est trouvée, nous générions son histogramme LBP et la colorions d'une couleur C ($C=marron$, dans l'exemple ci dessous). [Figure 2.3]



FIGURE 2.3 – Première détection.

Nous allons parcourir les carrés créés par l'algorithme de découpage en comparant leurs histogrammes à celui de la référence détectée.

Il existe plusieurs méthodes pour calculer la distance entre deux histogrammes. Nous utilisons la méthode d'intersection défini comme suit :

soit h_1 et h_2 les deux histogrammes à comparer. La valeur d de comparaison est égale à la somme des minimums des fréquences de h_1 et h_2 pour chacun de leurs éléments.

$$d(h_1, h_2) = \sum_i \min(h_1(i), h_2(i))$$

Plus les histogrammes se ressemblent, plus la valeur de d est grande. Son maximum vaut 1, si les histogrammes sont identiques.

Si la valeur de comparaison est au dessus du *seuil*, le carré correspondant dans l'IB est colorié de la même couleur C. [Figure 2.4]

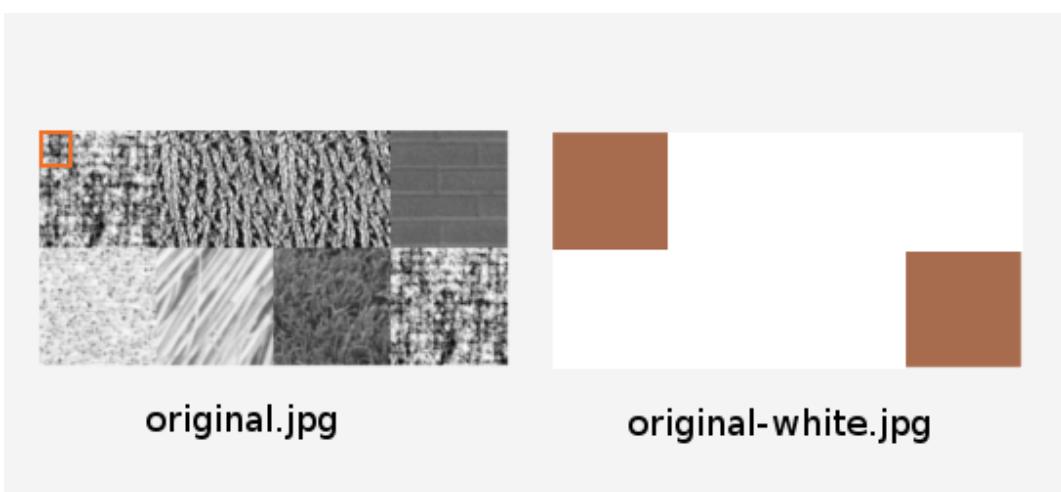


FIGURE 2.4 – Coloration de la première texture détectée.

La recherche d'une nouvelle référence est entamée. On continu à chercher le premier carré blanc avec la même taille de références t_r initialement fixée. [Figure 2.5]



FIGURE 2.5 – Deuxième détection.

Les carrés ressemblant à la nouvelle référence sont coloriés de la même façon. [Figure 2.6]

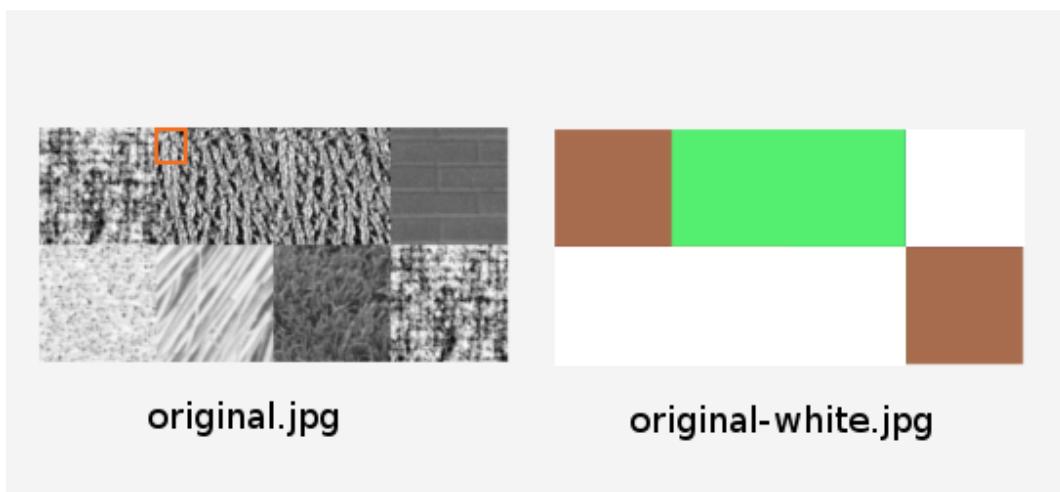


FIGURE 2.6 – Coloration de la deuxième texture détectée.

Si aucun carré blanc de taille t ne reste dans l'IB, l'algorithme est terminé. [Figure 2.7]

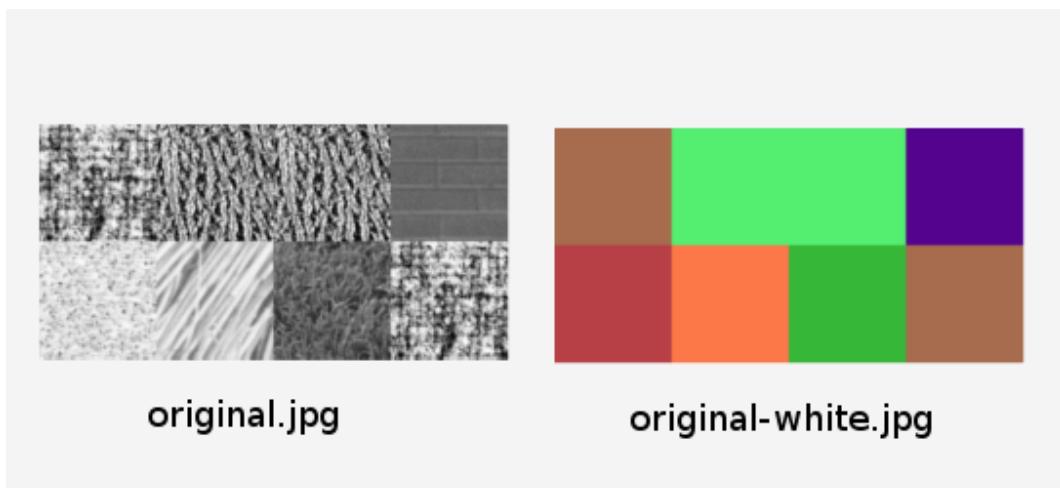


FIGURE 2.7 – Détection de toutes les textures.

2.4.3 Variantes

Le premier problème à traiter pour une meilleure précision est de trouver le bon *seuil* pour chaque texture recherchée. Et pour cela nous avons pensé à un algorithme d'apprentissage. Il s'agit d'une étape facultative. Cette dernière vise à tester d'une manière manuelle le meilleur *seuil* possible pour une texture donné, et le sauvegarder dans une base de données.

Quand vient l'étape de recherche de texture, les références seront recherchés dans la base de données suivant la ressemblance de l'histogramme sauvegardé et l'histogramme de la texture, et selon laquelle le meilleur *seuil* sauvegardé sera extrait.

LBP à rotation invariante

L'une des variantes les plus intéressantes de la LBP est la LBP^{riu2} qui est capable de détecter une texture même si cette dernière a subis une rotation d'un certain angle α qui est inclus dans l'ensemble suivant : $S = \{45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 270^\circ, 315^\circ\}$.

Dans la figure 2.8, après que l'image ait subis une rotation de 45° , on remarque bien que le motif a aussi subit une rotation de 45° , mais le plus important est que le nouveau motif fait partie aussi des «*uniform patterns*» tout comme le premier. Le motif a passé de la valeur $m = 11000011$ vers la valeur $m' = 11100001$.

De ce fait, on comprend qu'une rotation de l'image d'un angle de 45° fait décaler les valeurs des motifs de un chiffre vers la droite, de manière générale on a :

$$D = \frac{360^\circ}{\alpha}$$

La valeur du décalage D dépend de la valeur de l'angle de rotation α , et donc le choix de l'ensemble $S = \{45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 270^\circ, 315^\circ\}$ vient du fait que ce sont les seules valeurs de α qui donnent un nombre entier naturelle D de décalage.

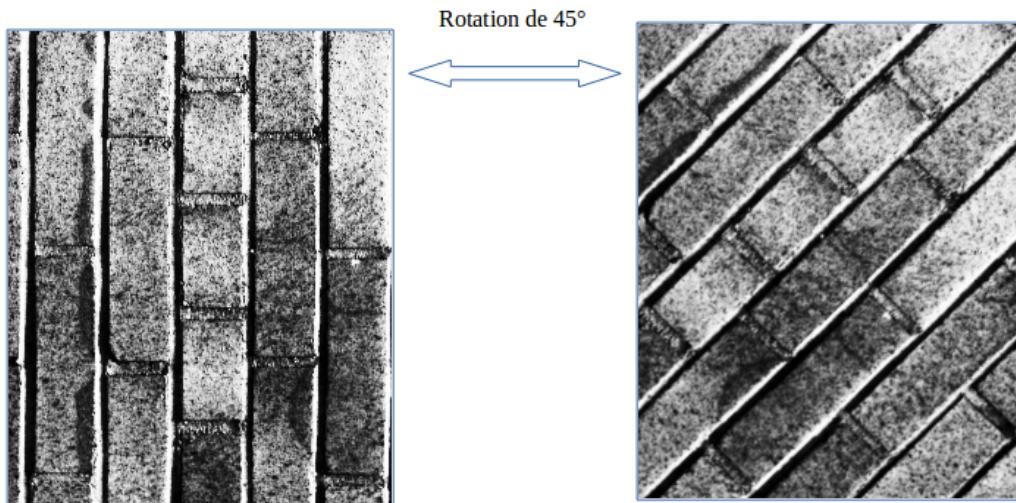


FIGURE 2.8 – Un motif après une rotation de 45°

Ainsi, comme chaque motif va subir un décalage de D positions, et que les huit motifs résultants des huit décalages possibles, peuvent représenter toujours la même texture mais qui est sous des angles α différents, on peut donc regrouper les huit rotations possibles de chaque motif dans une seule valeur de l'histogramme comme le montre la figure 2.9.

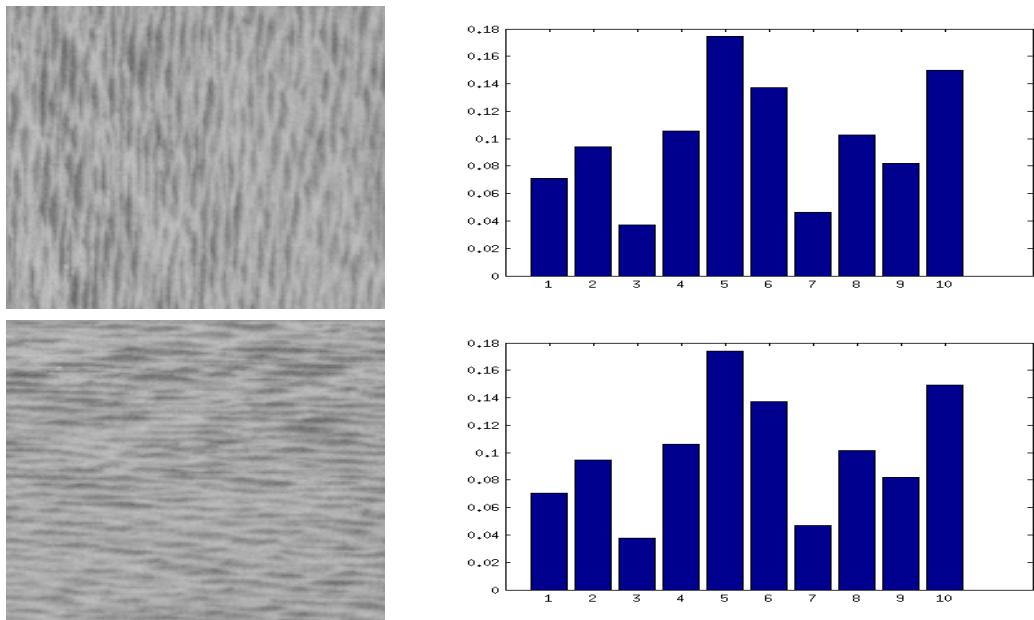


FIGURE 2.9 – Histogramme LBP^{riu^2} d'une texture subissant une rotation de 90°

L'histogramme LBP^{riu^2} a dix valeurs différentes qui sont :

- Deux valeurs pour les motifs $m = 00000000$ et $m = 11111111$.
- Sept valeurs pour les «uniform patterns» uniques (incluant leur huit rotations) qui sont : $m = 10000000$, $m = 11000000$, $m = 11100000$, $m = 11110000$, $m = 11111000$, $m = 11111100$, $m = 11111110$.
- Et une dernière valeur regroupant tous les «non uniform patterns».

On peut voir les meilleurs résultats obtenus dans la figure suivante :

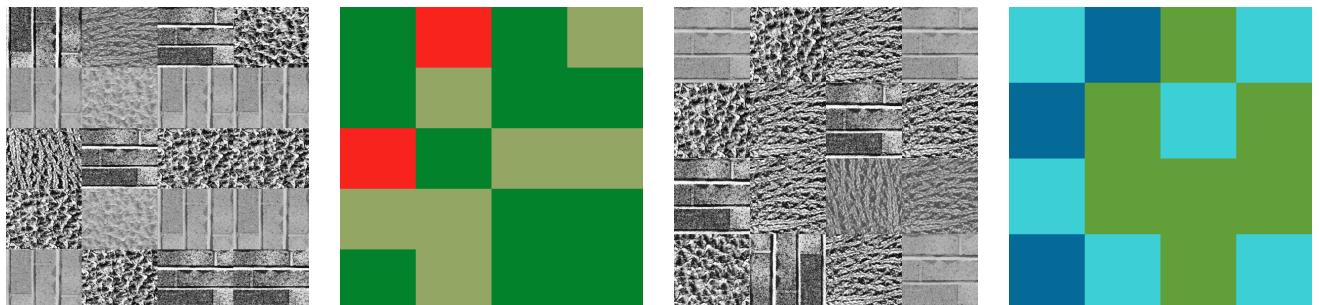


FIGURE 2.10 – Segmentation avec la LBP^{riu^2}

Cependant dans certains cas, les résultats sont moins bons et cela revient au manque de précision de l'histogramme LBP^{riu^2} , puisque chaque motif a été regroupé avec ses rotations, d'où résulte la difficulté du choix ... AJOUTER FIGURE HERE!!!!

Segmentation par des formes circulaires

En plus de la recherche par des carrés, nous avons implémenté d'autres formes géométrique pour améliorer la détection de ces derniers. Nous avons remarqué que, par exemple, les cercles n'étaient pas bien distingués par la méthode classique.

Prenons l'image représentée dans la figure 2.8.

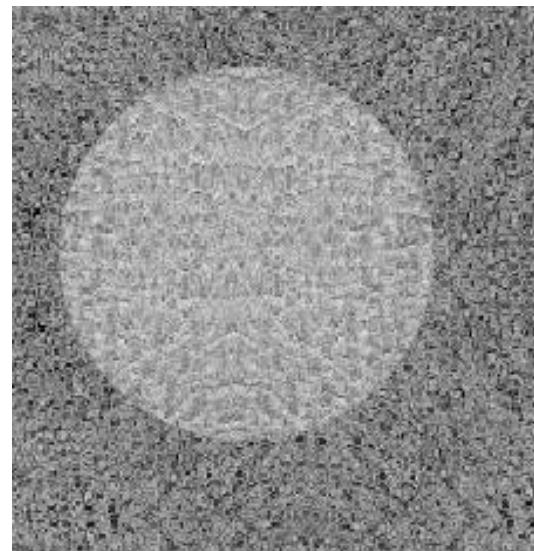


FIGURE 2.11 – Image initiale.

La méthode classique nous a donné comme résultat l'image représenté par la figure 2.9.

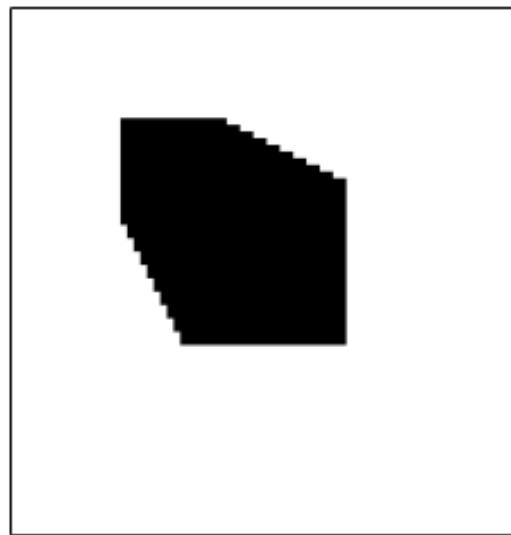


FIGURE 2.12 – Détection par carrés.

Nous avons changé la détection et le coloriage des carrés par une détection et un coloriage en cercle.

Nous parcourons le cercle pour créer l'histogramme et le colorier à l'aide de la formule suivante :

Prenons le centre $O(x_0, y_0)$ du cercle \mathbf{c} d'un rayon \mathbf{r} . Un pixel $P(x_p, y_p)$ fait partie du cercle si l'inéquation ci-dessous est réalisée :

$$(x_p - x_0)^2 + (y_p - y_0)^2 \leq R^2$$

Nous avons eu comme résultats l'image représentée par la figure 2.10.

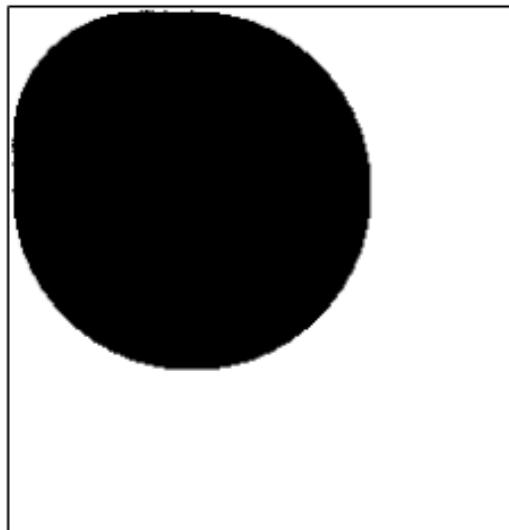


FIGURE 2.13 – Détection par cercles.

2.5 Création des images de test

Les images sur lesquelles nous avons appliqué nos tests ont été générées aléatoirement. Les textures ont été tirées de la base de donnée *Brodatz* [ref].

Nous avons utilisé un premier algorithme pour découper ces dernières en plus petite textures (de taille 150*150px) et les avons mis dans un dossier *Textures*.

```

Function Textures' Cutting(Brodatz textures : folder) : folder
  For Each image from Brodatz's textures do
    | - Cut 150x150 first pixels.
    | - Add the new texture to the Textures folder
  end For
  return Textures
End
```

Algorithm 3: Cutting textures

Puis nous avons élaboré un algorithme qui sélectionne aléatoirement des textures à partir du dossier rempli précédemment et construit avec ces dernières une image à tester.

```

Function Create Test Image(Textures : folder) : image
    Create a matrix M of random dimensions For each cell in M do
        | Affect a random texture from Textures
    end For
    image = transforming M into one image
    return image
End

```

Algorithm 4: Création de l'image à traiter

2.6 Conclusion

Nous avons présenté dans ce chapitre les détails de notre approche dans le but de segmenter les différents types de textures. Les techniques que nous avons utilisé et généralisé nous ont permis d'élaborer un programme qui détecte toutes les textures présentes sur une image quelconque, et cela d'une manière automatique.

Les résultats des tests restent satisfaisante si nous respectons les contraintes cités au début de ce chapitre, et qui concernent la forme rectangulaire des textures recherchées. Nous avons aussi montré que d'autres formes de textures peuvent être appliquées à notre méthode afin de détecter des formes non rectangulaires.

Nous présentons dans ce qui suit, notre application ainsi que des tests qui montreront les différents résultats obtenu à l'aide de cette méthode.

Chapitre 3

Application et tests

3.1 Introduction

Nous présentons dans ce chapitre les résultats de notre application et leur interprétation. Ainsi que les détails des fonctionnalités disponibles.

3.2 Outils utilisés

Nous avons choisi comme langage de programmation le langage Python2.7 et ce pour diverses raisons dont on site [HOY 10] :

- Facilité d'écriture et de compréhension du code. On trouve moins de barrière entre la génération d'idée, et leurs traduction en code.
- Balance entre langage de haut et de bas niveau. Ce qui forme un point fort du langage.
- L'interopérabilité avec les autres langages, ou la capacité très utile à pouvoir appeler des script d'autres langages facilement.
- Le grand nombre de bibliothèques disponibles.
- La documentation abondante.

Le traitement bas niveau des images se fait avec la bibliothèque graphique libre OpenCV qui propose un bon nombre de fonctions permettant d'effectuer les opérations classiques.

Nous avons choisi d'utiliser pour le développement de l'interface graphique le framework PyQt, un module qui permet de lier le langage Python avec la bibliothèque Qt. En utilisant l'environnement *Qt4 Designer version 4.8.1*.

Pour améliorer la vitesse de découpage, il s'est avéré qu'appeler un script matlab est bien plus intéressant en terme de temps d'exécution que de reprogrammer le script en python. La version de matlab utilisée étant *R2012a 7.14.0.739 32-bit(glnx86)*

Ce document est rédigé en *Latex*, avec *Texmaker 3.2*.

3.3 Presentation et test de l'application :

Nous commençons par exécuter l'application. [Figure 3.1]

Elle nous permet de choisir une image en cliquant sur Browse pour parcourir nos fichiers, ou en collant directement son chemin absolu.

Nous choisissons ensuite la valeur du *threshold*.

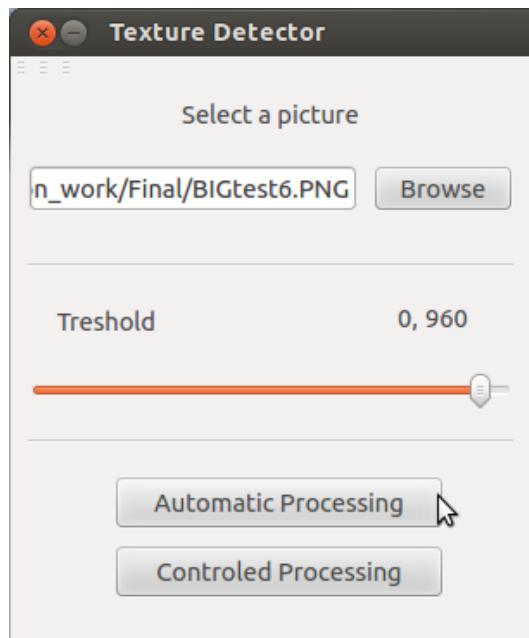


FIGURE 3.1 – Fenêtre principale.

Nous avons après cela le choix entre deux méthodes :

- Automatic Processing : qui fait un traitement automatique des textures
- Controlled Processing : qui permet d'avoir plus de contrôle sur le traitement et la reconnaissance des textures. Il est aussi considéré comme processus d'entraînement de l'analyseur.

Nous commençons par le traitement automatique. La fenêtre de la figure 3.2 nous est affiché, elle montre à droite l'image sélectionnée et en bas le *threshold*, avec possibilité de les modifier.

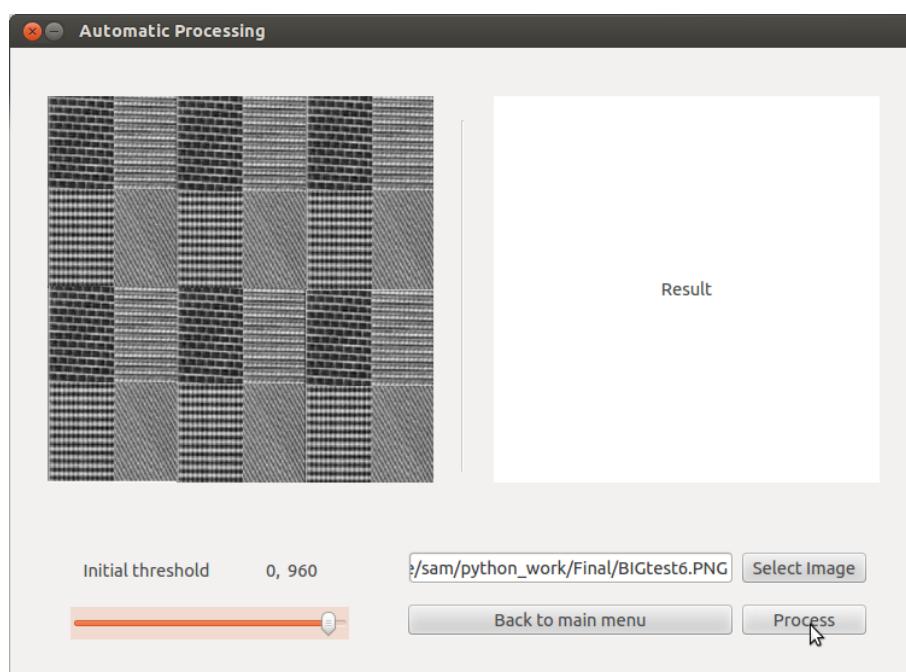


FIGURE 3.2 – Traitement automatique.

Nous cliquons sur *Process* pour lancer le traitement. Après quelques instants, le résultat

de la détection et coloration des textures nous est affiché à droite de la fenêtre. [Figure 3.3]

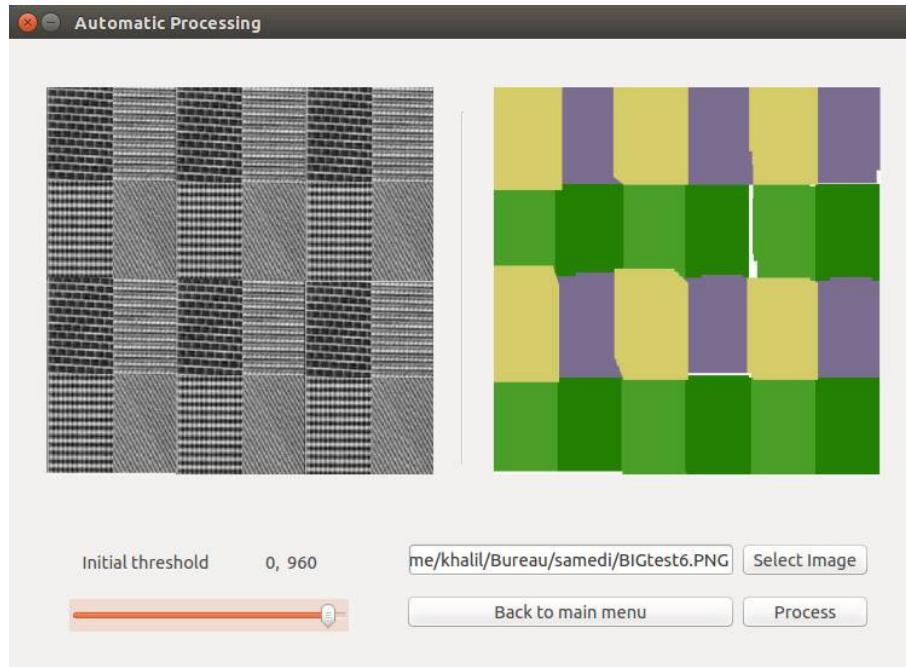


FIGURE 3.3 – Traitement automatique fini.

L'image résultante est automatiquement enregistrée dans le dossier contenant l'application. Comme expliqué dans le précédent chapitre, les *thresholds* utilisés pour les différentes textures détectées sont tirés d'une base de données remplit à chaque usage du traitement contrôlé.

Nous revenons vers le menu principale en cliquant sur le bouton *Back to main menu*, et nous choisissons cette fois *Controlled Processing*. Nous allons tester le traitement contrôlé où nous allons pouvoir modifier à chaque fois le *threshold* de chaque texture détectée. Nous changerons aussi les informations présente dans la base de donnée en sauvegardant les résultats satisfaisants des détections d'une texture x.

Prenons l'exemple de la figure 3.4. La première étape étant identique au traitement automatique, nous choisissons l'image et un *threshold* par défaut. Nous commençons le traitement en cliquant sur *Proces*.

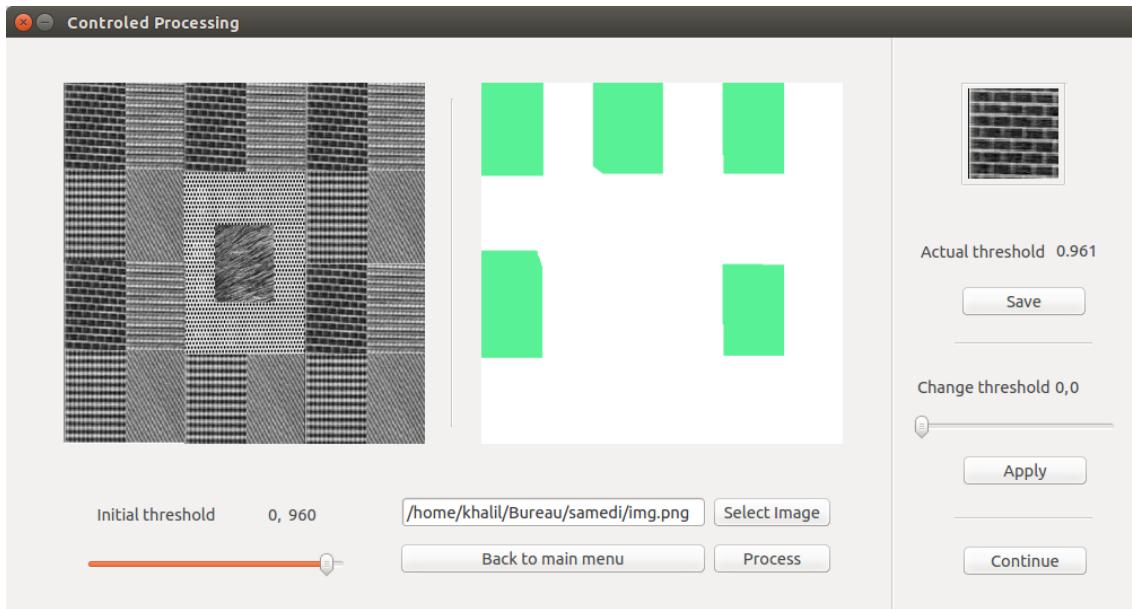


FIGURE 3.4 – Traitement contrôlé - Itération 1.

Une première texture est détectée, et les régions ressemblantes à cette dernière sont colorées. Remarquons que le *threshold* utilisé (*Actual threshold 0.961*) est différent de celui choisi pour être par défaut (*Initial threshold 0.960*). Ceci est du au fait que la même texture, ou une texture fortement ressemblante ai été trouvé dans la base de donnée. Le *threshold* de cette dernière a été donc choisi.

Nous avons, ensuite, la main pour choisir de sauvegarder le *threshold* dans la base de donnée, de continuer si le résultat est satisfaisant, ou de changer ce premier et rechercher les zones ressemblantes à la texture détectée à nouveau.

Nous choisissons de continuer, jusqu'a ce que l'a détection de l'une des textures n'est pas satisfaisante. Nous choisissons dans ce cas de changer le *threshold*.[Figure 3.5]

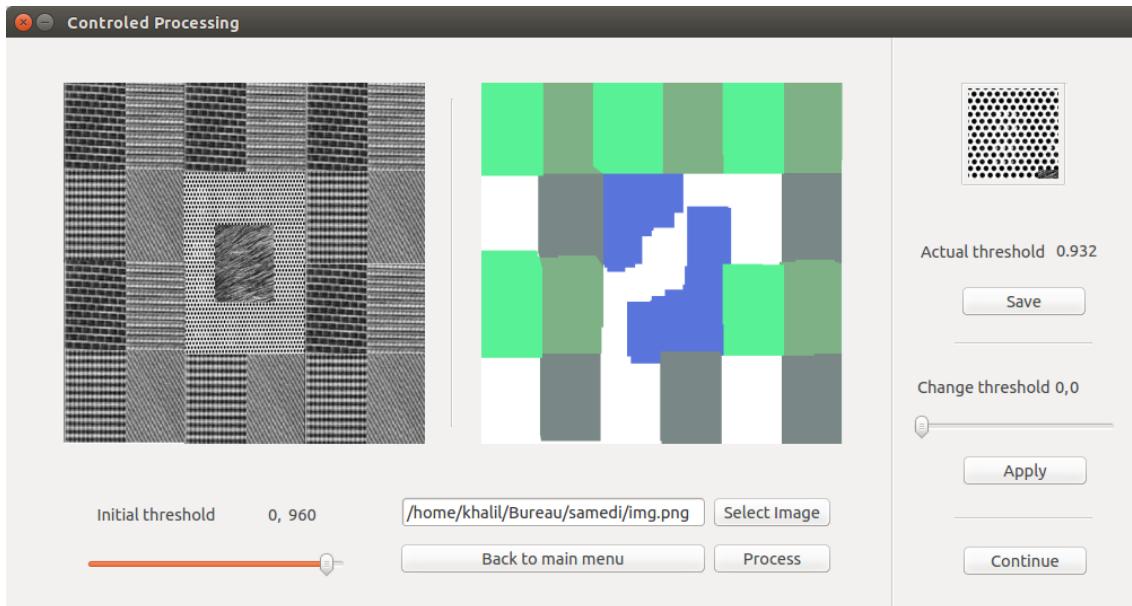


FIGURE 3.5 – Traitement contrôlé - Itération 4.

Nous cliquons sur apply pour cela après avoir choisi un nouveau *threshold* dans la partie *Change threshold* [Figure 3.6].

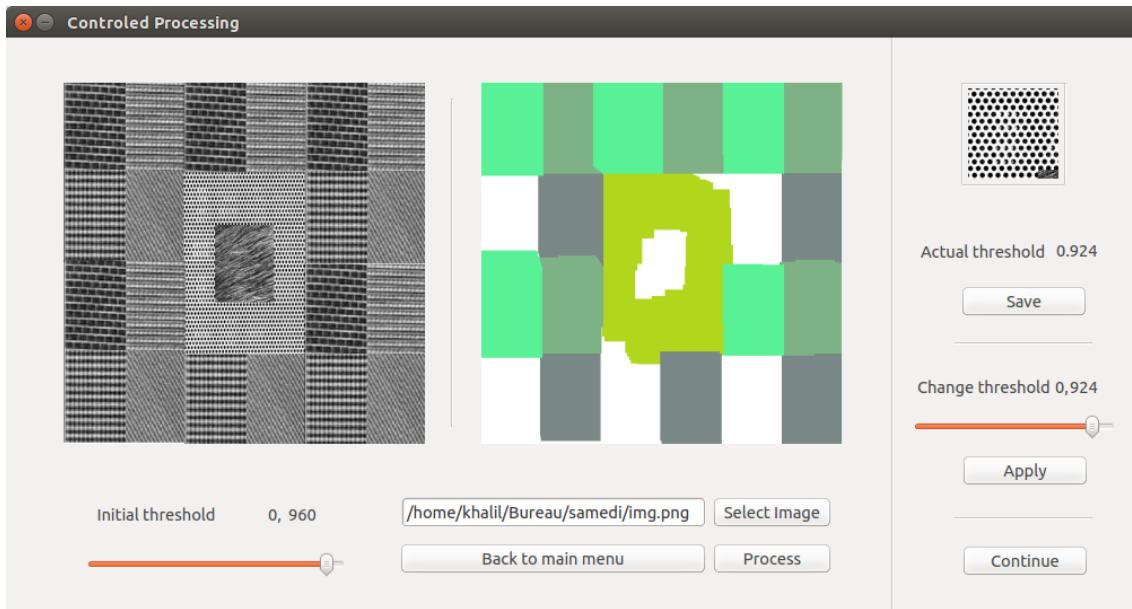


FIGURE 3.6 – Traitement contrôlé - Itération 5.

Nous allons sauvegarder ce *threshold* dans la base de donnée en appuyant sur le bouton *Save* pour que la prochaine fois cette valeur sera utilisé automatiquement.

Le processus continue de la sorte jusqu'à ce que toutes les textures de l'image soient détectées [Figure 3.7].

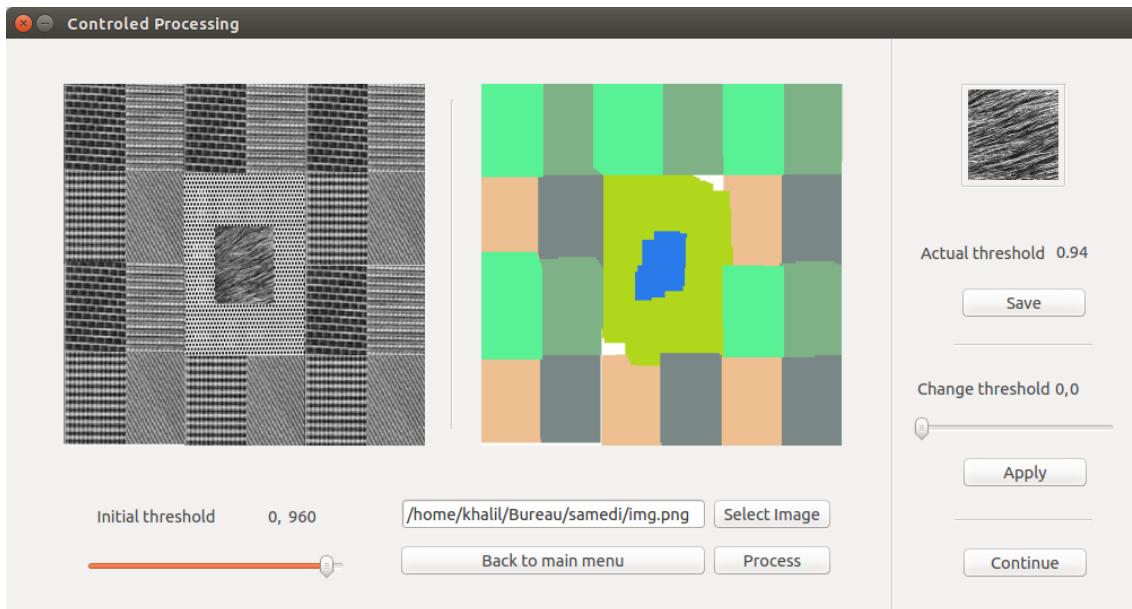


FIGURE 3.7 – Traitement contrôlé - Itération 7.

3.4 Tests et interprétation des résultats :

Nous allons voir plus en détails les résultats de plusieurs tests pour voir le comportement du threshold vis-à-vis des différentes textures.

Les détails des threshold choisi pour les textures détectés et leur surface occupé dans l'image (par carré de 150*150 px) sont représentés dans le tableau suivant .

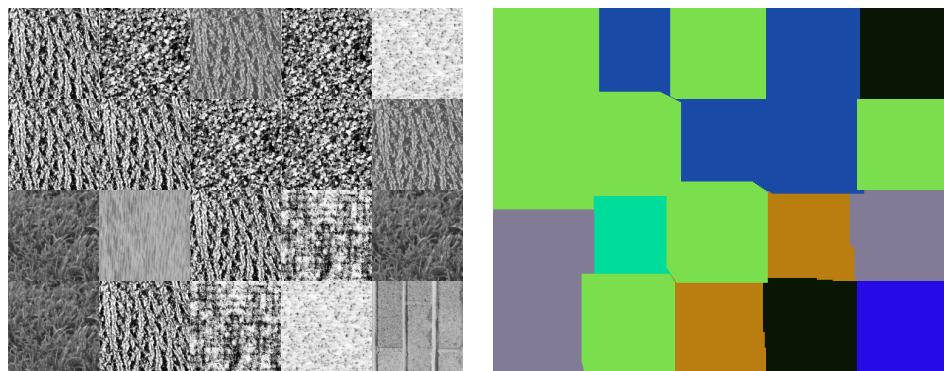


FIGURE 3.8 – Test 1

texture	surface (par carré)	threshold
1	2	0,969
2	3	0.997
3	4	0.973
4	2	0.971
4	3	0.964
6	1	0.964

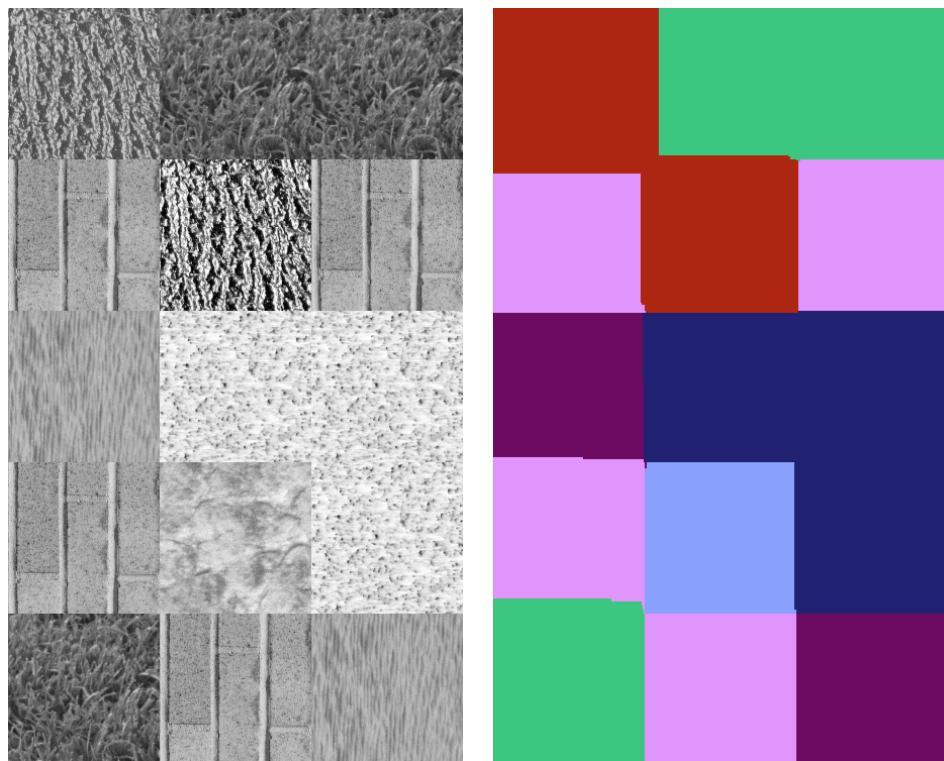


FIGURE 3.9 – Test 2

texture	surface (par carré)	threshold
1	7	0,972
2	4	0.971
3	2	0.969
4	3	0.970
5	1	0.976
6	2	0.958
7	2	0.973

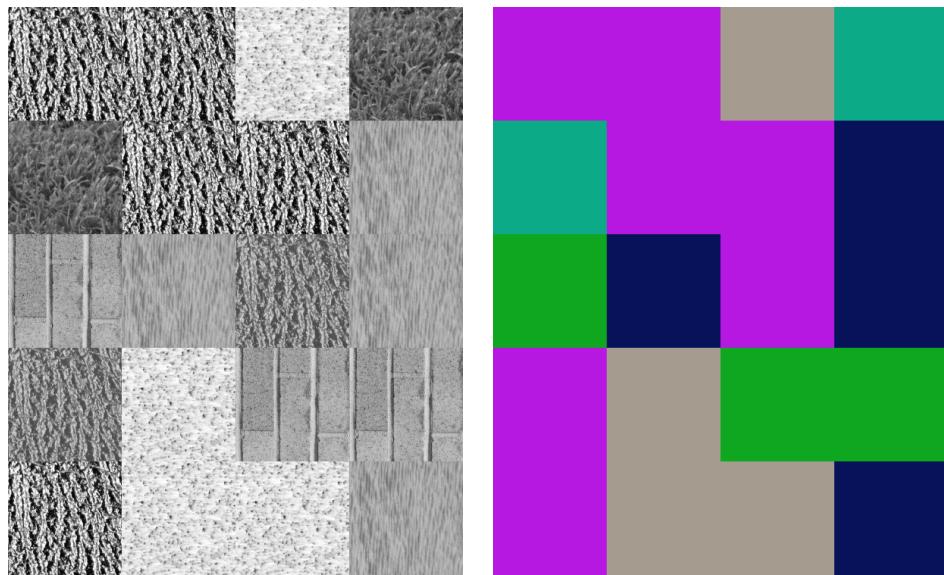


FIGURE 3.10 – Test 3

texture	surface (par carré)	threshold
1	7	0,972
2	4	0.972
3	2	0.965
4	4	0.958
5	3	0.973

Nous avons fait une dizaine de tests similaires. Nous avons calculé les moyenne des threshold par image dans un premier lieu, et puis, nous avons pris en compte les coefficients en fonction de la surface qu'ils occupent dans l'image.

Les résultats sont représentés dans le graphe suivant :
Graph représentatifs des moyennes.

Nous remarquons que, malgré la différence considérable entre les thresholds des différentes textures, la moyenne entre ces derniers restes constante. Et ce, que ce soit avec ou sans compter le coefficient relative à la surface occupée par les textures.

Ceci implique que la précision en terme de détection reste élevée.

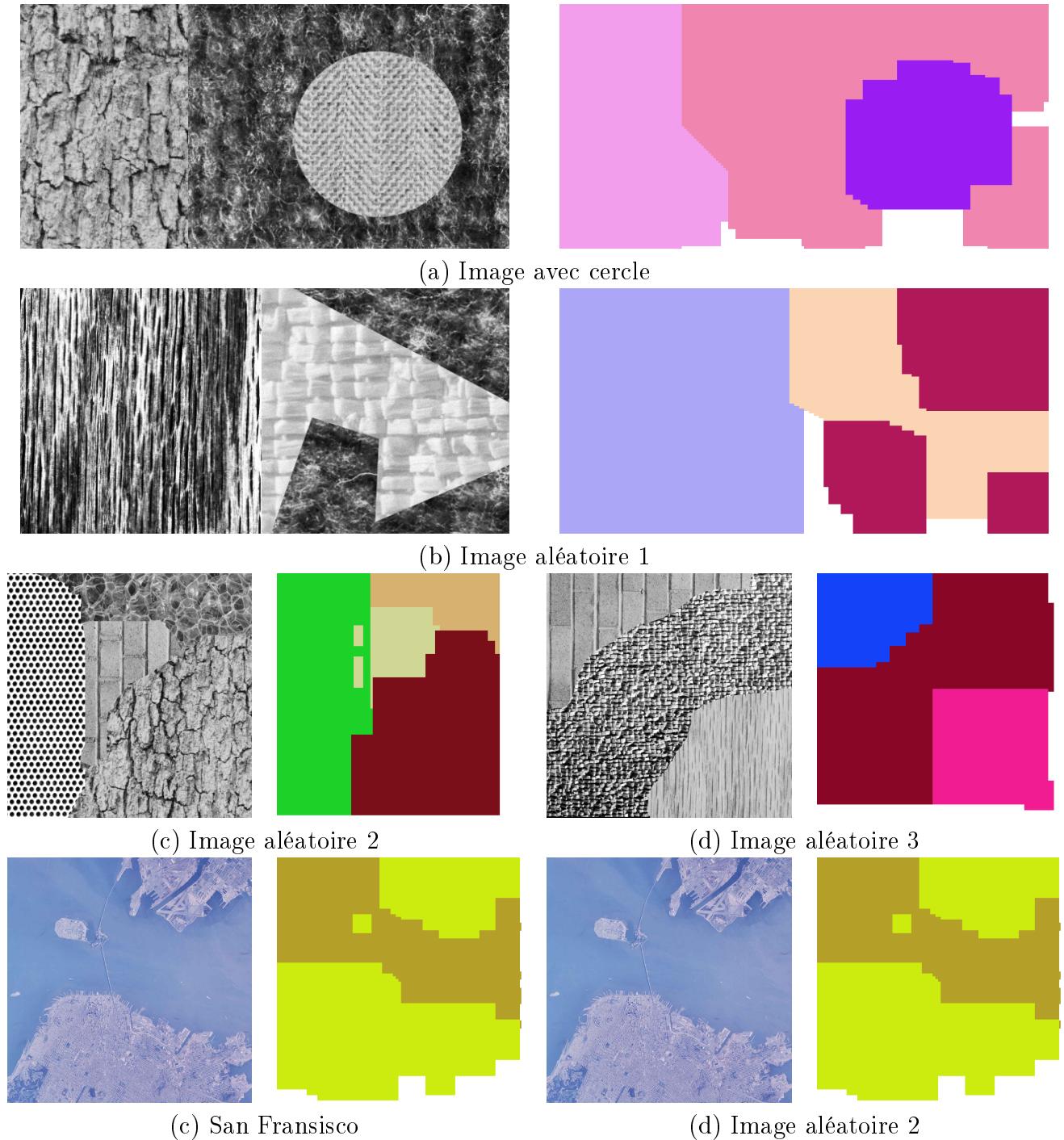


FIGURE 3.11 – La segmentation d’images

3.4.1 Tests supplémentaires

3.5 Conclusion

Dans ce dernier chapitre nous avons décrit l'interface de notre application et les détails de son fonctionnement. Nous avons fait une démonstration des deux différents modes d'utilisation : **automatique** et **contrôlé** (étape d'apprentissage) et avons aussi exposé les multiples options proposées dans ce dernier mode.

Nous avons présenté une série de tests effectués sur différentes images aléatoirement construites et qui donnent des résultats satisfaisants et très prometteurs.

Enfin, nous avons vu l'indépendance de cette méthode des formes utilisées pour la recherche et la reconnaissance des textures, qui représente un très grand avantage.

Conclusion générale

Nous avons parcouru, tout au long de ce mémoire, les différentes étapes de la segmentation d'image basées sur la méthode de reconnaissance de texture, et nous pouvons dire que ce travail démontre la puissance de cette dernière.

Nous avons étudié les aspects statistiques de la caractérisation des textures, et plus précisément, la caractéristique LBP qui nous a offert un outils de reconnaissance très puissant. Nous avons aussi étudié la méthode de segmentation dynamique qui permet de détecter une texture donnée, dans une image donnée. Notre travail consistait à élaborer une méthode de généralisation pour la détection automatique de toutes les textures présentes dans une image, sans avoir, au préalable, aucune information à part l'image elle même. Les résultats obtenu restent très prometteurs et motivants pour accorder à cette étude plus de temps et d'efforts.

Bien que certaines contraintes ont du être respectés, il faut savoir que ce travail est une première étape d'application de cette méthode. Némoins, nous aurions souhaité bénéficier de plus de temps pour mettre en application nos idées qui consistent à, premièrement, lier la reconnaissance et la segmentation de texture à des descriptions dans un format compréhensible par une machine. Celle-ci, pourrait très bien comprendre si elle est en train de survoler une zone forestière ou urbaine.

Aussi, pour trouver un accord entre les différentes formes géométriques pour la détection des textures, nous pensons à continuer avec une reconnaissance de formes régulières. Ce travail peut-être enrichi en implémentant l'algorithme avec des formes hexagonales. Ces formes représentent, théoriquement, un très bon compromis entre les différentes autres formes. Leur nature permet d'englober uniformément une surface entière tout en évitant les chevauchements (contrairement aux cercles, par exemple). Son avantage par rapport aux formes carrées et aux triangles équilatérales est le fait qu'un hexagone couvre bien plus efficacement une surface ronde. En effet, le rapport entre la surface d'un triangle équilatéral et un cercle est de 17.77%. Celle d'un carré est de 63.7%. Enfin, celle d'un hexagone n'est pas moins de 83%.

Nous avons décidé de continuer personnellement ce travail afin d'aboutir à d'autres résultats souhaités.