

Traitement d'image

TP n°08

BARCHID Sami

CARTON Floriane

Introduction

La matière vue dans ce TP traite du tatouage d'une image par étalement de spectre. L'objectif ici sera de compléter et analyser un système de tatouage qui permet l'insertion (embedding) d'un message de N bits dans une image hôte en niveaux de gris et qui permet le décodage d'un message caché dans une image déjà tatouée.

Ce rapport de TP est divisé en trois parties :

- **Filtres de contours** : mise en œuvre du calcul de la norme du gradient de Sobel afin de comprendre la notion de détection de contours qui en découle.
- **Tatouage par étalement de spectre** : complétion du système de tatouage par étalement de spectre puis analyse et interprétation des résultats obtenus.
- **Robustesse du schéma de tatouage** : Analyses et interprétations du décodage d'images tatouées par étalement de spectre ayant subis diverses transformations afin de comprendre la robustesse des images tatouées.

Filtres de contours

Dans cette partie du TP, nous allons comprendre la méthode pour déterminer les contours à partir de la norme du vecteur gradient sur « *peppers* » l'image suivante :



IMAGE - « *peppers* »

Il faut ensuite calculer $\frac{\partial f}{\partial x}$ et $\frac{\partial f}{\partial y}$, les composantes du gradient selon l'axe vertical et horizontal. Le calcul de ces dérivées partielles selon les directions du plan est réalisé grâce à l'opération de convolution par les filtres de Sobel.

Pour ce faire, il faut tout d'abord convertir l'image « *peppers* » sur 32 bits. Cette conversion est nécessaire car, comme nous travaillons sur une

dérivation, nous obtenons des valeurs négatives et positives par rapport aux contours. Les filtres de convolution de Sobel se trouvent ci-dessous.

$$H_x = \begin{bmatrix} -0.125 & 0 & 0.125 \\ -0.25 & 0 & 0.25 \\ -0.125 & 0 & 0.125 \end{bmatrix}, H_y = \begin{bmatrix} -0.125 & -0.25 & -0.125 \\ 0 & 0 & 0 \\ 0.125 & 0.25 & 0.125 \end{bmatrix}$$

Les images filtrées de Sobel obtenues sont les suivantes :

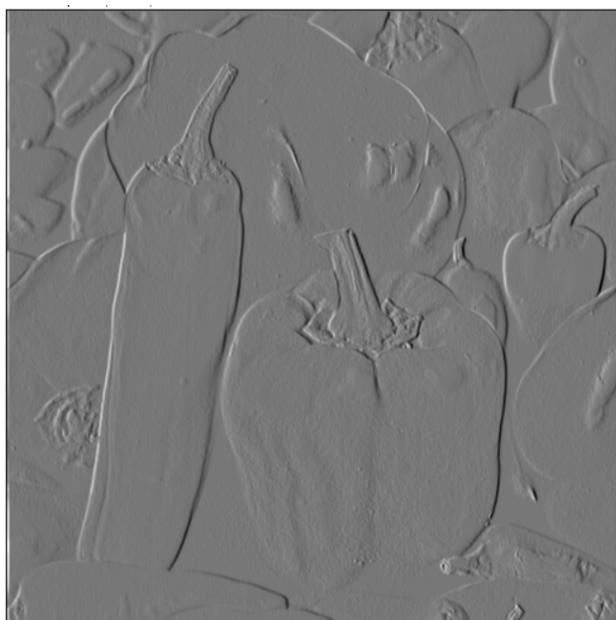


IMAGE - $\frac{\partial f}{\partial x}$

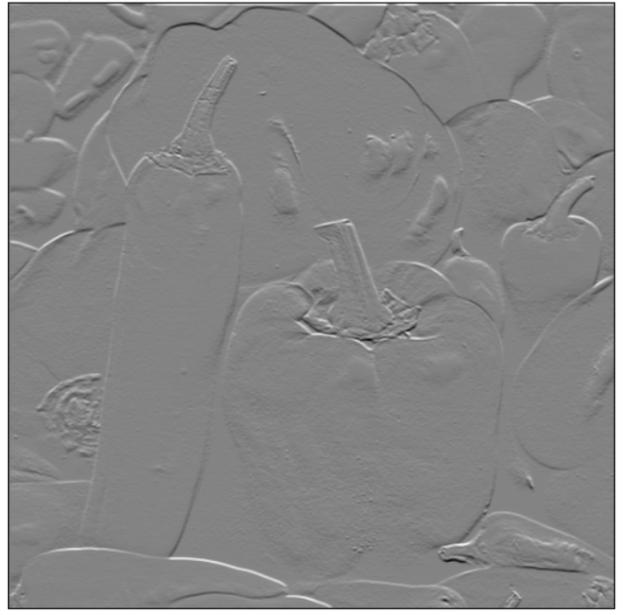


IMAGE - $\frac{\partial f}{\partial y}$

Dans ces deux images converties sur 32 bits, on retrouve :

- Des pixels très sombres et très clairs qui définissent les contours selon l'axe horizontal pour $\frac{\partial f}{\partial x}$ et selon l'axe vertical pour $\frac{\partial f}{\partial y}$.

Sur 32 bits, ces pixels ont des valeurs négatives (sombres) et fortement positives (clairs).

- Des pixels gris (donc des valeurs nulles sur 32 bits) qui représentent les parties de l'image qui ne sont pas des contours.

Grâce à ces deux images nous pouvons alors calculer en chaque pixel la norme du gradient en appliquant la formule :

$$|\vec{\nabla} f| = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2}$$

En seuillant l'image obtenue par la norme du gradient, nous pouvons afficher un contour :



IMAGE – Norme du gradient seuillée

Le seuil utilisé est une valeur de niveau de pixel = 12,71, ce qui garde 11.26 % des pixels de l'image.

Il est donc possible de trouver un seuil global permettant de mettre en évidence les pixels contours de manière satisfaisante.

Tatouage par étalement de spectre

Dans cette partie du TP, nous allons compléter l'algorithme de tatouage par étalement de spectre pour un message de 16 bits.

La différence entre le fonctionnement du système de tatouage par étalement que l'on va compléter et celui vu en cours théorique est que, ici, chaque porteuse va tatouer le message entier sur sa tuile. Contrairement à ce qui a été vu en cours où une porteuse tatoue un seul bit du message sur la tuile.

Les paramètres que nous pouvons changer dans ce système de tatouage par étalement de spectre sont :

- L'image à tatouer
- Le PSNR cible qui va influencer le facteur gamma, c'est-à-dire que l'on définit un PSNR cible pour maîtriser la distorsion
- La clé privée (seed) qui permet de tatouer une image et/ou de retrouver le message sur une image tatouée avec la même clé privée
- Le message que l'on veut insérer dans l'image tatouée

Compréhension de la création du signal de tatouage

Pour comprendre l'élaboration du signal de tatouage, nous allons analyser les instructions implémentées dans le langage macro ImageJ : (*note : des commentaires ont été posés afin de détailler les instructions*)

```

280 /**
281 * Paramètres :
282 *      IDCARRIERS sont les porteuses (chacun contient le message)
283 *      widthImg est la largeur de l'image à watermarker
284 *      heightImg est la hauteur de l'image à watermarker
285 *      msg est le message qui doit être placer dans le tatouage (tableau de bits)
286 */
287 function makeWmkSignal(IDcarriers,widthImg,heightImg,msg)
288 {
289     selectImage(IDcarriers);
290     getDimensions(widthTile, heightTile, channels, Nc, frames);
291
292     // Création d'une image noire qui sera le signal de tatouage
293     newImage("WmkSignal", "32-bit black", widthImg, heightImg, 1);
294     IDWmk = getImageID();
295
296     // Création d'une image qui deviendra l'image concaténée des tuiles
297     newImage("WmkTile", "32-bit black", widthTile, heightTile, 1);
298     IDWmkTile = getImageID();
299
300
301     //Application de la formule du vecteur de tatouage
302     /* Spread-Spectrum */
303     for(i=0; i< Nc; i++)
304     {
305         selectImage(IDcarriers);
306         //indexes of stacks start at 1!!
307         setSlice(i+1);
308
309         if (msg[i] == 1)
310             run("Multiply...", "value=-1 slice");
311         imageCalculator("Add 32-bit", IDWmkTile, IDcarriers);
312     }
313
314     selectImage(IDWmkTile);
315
316     /* update display LUT */
317     resetMinAndMax();
318
319     // Copier la tuile calculée
320     makeRectangle(0, 0, widthTile, heightTile);
321     run("Copy");
322     run("Select None");
323
324     // Répéter la tuile calculée plusieurs fois sur toute l'image
325     // et la concatener pour obtenir le signal de tatouage
326     selectImage(IDWmk);
327     for(j = 0 ; j < widthImg; j = j + widthTile)
328     {
329         for(i = 0 ; i < heightImg; i = i + heightTile)
330         {
331             makeRectangle(j, i, widthTile, heightTile);
332             run("Paste");
333         }
334     }
335     run("Select None");
336     return IDWmk;
337 }

```

En bref, cette fonction calcule la construction du vecteur de tatouage \mathbf{w} . La formule de la construction du vecteur de tatouage vu en cours était la suivante :

$$\mathbf{w} = \sum_{i=0}^{N_c-1} (-1)^{\mathbf{m}(i)} \mathbf{u}_i$$

avec :

- \mathbf{w} est le vecteur de tatouage
- N_c est le nombre de bits du message
- \mathbf{u}_i est la porteuse i
- $\mathbf{m}(i)$ est le bit i du message \mathbf{m}

La différence avec la formule vue en cours est que, ici, les porteuses contiennent chacune le message en entier. De ce fait, chaque tuile contient le message entier.

Ainsi, lorsque l'on décode une image tatouée, on vérifie pour chaque bit du message que la majorité des tuiles ont la même valeur pour ce bit. L'intérêt est que si l'image tatouée subit une transformation, il y a peu de chances que la majorité des tuiles aient subies une transformation pour le même bit. On augmente donc en robustesse mais on diminue en sécurité puisqu'on n'a besoin que d'une partie de l'image pour décoder tout le message de l'image.

Nous calculons donc la tuile pour le vecteur de tatouage et l'image du signal de tatouage devient une répétition de cette tuile.

Implémentation de calcul du bit error rate (BER)

Pour mesurer le ratio de bits mal décodés lorsque l'on décode une image tatouée (et ainsi pouvoir mesurer la robustesse du tatouage), il faut calculer le bit error rate (ou « BER »), défini par la formule suivante :

$$BER = \frac{d_H(\hat{\mathbf{m}}, \mathbf{m})}{N_c}$$

Avec :

- **BER** est le bit error rate
- m est le message original
- \hat{m} est le message décodé
- $d_H(m, \hat{m})$ est la distance de hamming entre le message et son décodage, à savoir que le nombre de bits qui diffèrent entre le message et le message décodé

L'implémentation dans le langage de macro ImageJ est la suivante :

```
235 /**
236 * Calcul le BER
237 */
238 function bitErrorRate(Nc, msg, msgDecode) {
239     // calcul de la distance de hamming (le nombre de bits différents
240     // entre les deux messages)
241     distanceHamming = 0;
242     for (i = 0; i < Nc; i++) {
243         if(msg[i] != msgDecode[i]) {
244             distanceHamming++;
245         }
246     }
247     return distanceHamming / Nc;
248 }
249
250
```

Présentation de l'image de test

Les analyses des expériences sur le tatouage par étalement de spectre implémenté seront testées sur l'image suivante, nommée « *JUL* » :



IMAGE - « *JUL* »

Nous ferons également varier le paramètre du PSNR utilisé pour maîtriser la distorsion du tatouage en comparant les résultats pour trois valeurs de PSNR différentes :

- $PSNR_1 = 20 \text{ dB}$
- $PSNR_2 = 35 \text{ dB}$
- $PSNR_3 = 40 \text{ dB}$

Application du tatouage

Nous allons appliquer sur « *JUL* » le tatouage par étalement de spectre en utilisant les PSNR cibles définis.

Le tatouage par étalement implémenté calcule le signal de tatouage pour « *JUL* » suivant :

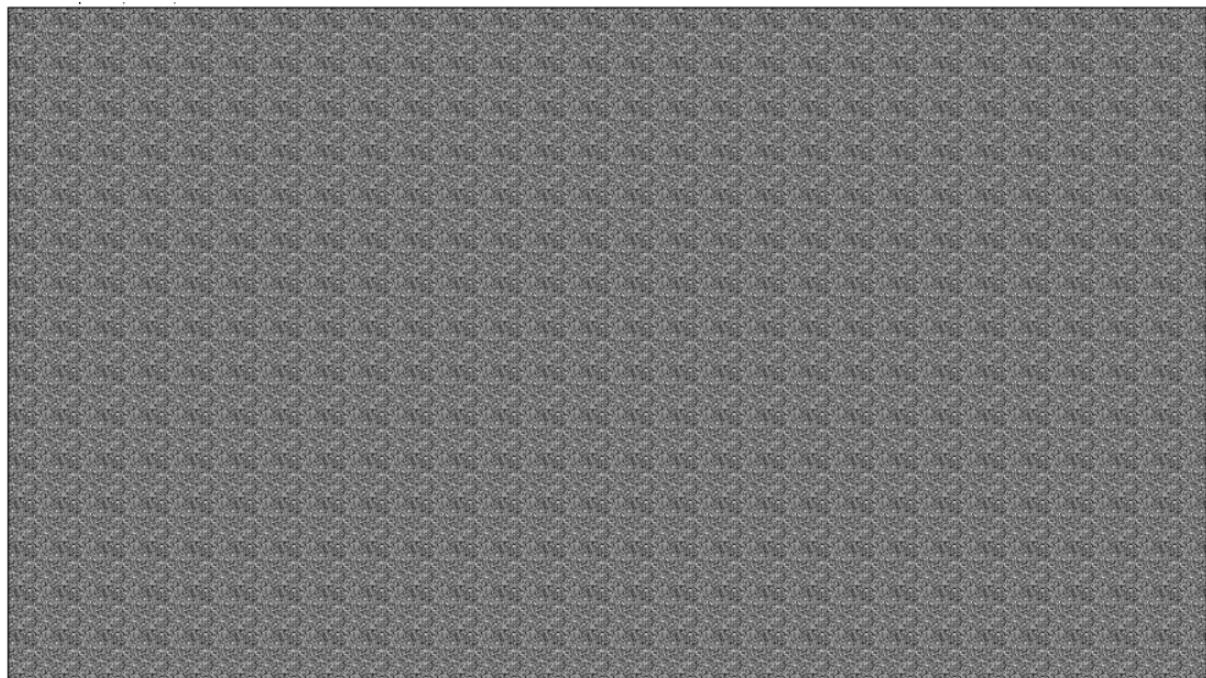


IMAGE - Signal de tatouage de « *JUL* »

PSNR cible	Image tatouée obtenue
$PSNR_1$	 <p data-bbox="822 1042 953 1080">BER = 0</p>

$PSNR_2$



BER = 0

$PSNR_3$



BER = 0

On remarque à l'œil nu que l'image du signal de tatouage a été additionnée à l'image « JUL ». Cette addition entre les deux pixels est

limitée par le PSNR cible choisi. En effet, on voit bien pour l'image tatouée grâce au PSNR cible $PSNR_1$ que l'image du signal de tatouage est bien présente.

Pour faire à nouveau le parallèle avec le cours théorique, l'image tatouée \mathbf{y} est obtenue par la formule suivante :

$$\mathbf{y} = \mathbf{x} + \gamma \mathbf{w}$$

où

- \mathbf{x} est l'image hôte (ou vecteur hôte), l'image d'origine
- \mathbf{w} est le signal de tatouage (ou vecteur de tatouage)
- γ est le coefficient qui dépend du PSNR cible et qui sert à limiter la distorsion du tatouage

On peut noter que pour le PSNR cible $PSNR_3=40 dB$, le signal du tatouage devient imperceptible.

D'autre part, le décodage des trois images tatouées obtenues donnent toutes un $BER=0$, ce qui prouve que le message peut être décodé sans problème dans l'image.

Exploitation du système visuel humain (SVH)

Nous allons exploiter les propriétés du SVH (nos yeux, en gros) pour améliorer l'insertion du signal de tatouage dans l'image.

En résumé, le but est de limiter le signal du tatouage dans les zones à basses fréquences et de l'augmenter dans les zones à hautes fréquences (donc les contours). Ceci vient du fait que l'œil humain est moins sensibles aux contours etc.

C'est pour cela que nous allons prendre en compte le SVH pour l'insertion du signal de tatouage en utilisant le gradient de Sobel (qui permet de détecter les contours) : pour chaque pixel de coordonnées (x, y) on multiplie le signal de tatouage par la norme du gradient de Sobel en ces coordonnées.

L'implémentation de cette opération dans le langage macro d'ImageJ est décrite ci-dessous :

```
109     // Calculer la norme du gradient de Sobel pour l'image hôte
110     normGrad = makeGradientNorm(IDhost);
111
112     // POUR CHAQUE [pixel (x,y)]
113     for (x = 0; x < widthImg; x++) {
114         for (y = 0; y < heightImg; y++) {
115             selectImage(normGrad);
116             gradPx = getPixel(x, y);
117
118             selectImage(IDwmk);
119             pix = getPixel(x, y);
120
121             // Multiplier le signal de tatouage par le gradient de Sobel
122             // pour ce pixel de coordonnée (x,y)
123             setPixel(x, y, gradPx * pix); |
124         }
125     }
126
127     // N.B. : on peut aussi utiliser l'image calculator
128     //imageCalculator("Multiply", IDwmk, normGrad);
```

Et la fonction du calcul de la norme du gradient de Sobel pour une image est donnée ci-dessous:

```
251 function makeGradientNorm(image) {  
252     //Duplication de l'image pour obtenir dx et dy  
253     selectImage(image);  
254     run("Duplicate...", "sobelX");  
255     sobelX = getImageID();  
256  
257     run("Duplicate...", "sobelY");  
258     sobelY = getImageID();  
259  
260     //Calcul des convolution par filtre de Sobel  
261     selectImage(sobelY);  
262     run("Convolve...", "text1=[-0.125 -0.25 -0.125\n0 0 0\n0.125 0.25 0.125] normalize");  
263     run("Square");  
264  
265     selectImage(sobelX);  
266     run("Convolve...", "text1=[-0.125 0 0.125\n-0.25 0 0.25\n-0.125 0 0.125] normalize");  
267     run("Square");  
268  
269     //sobelX2 + sobelY2  
270     imageCalculator("Add 32-bit", sobelX, sobelY);  
271  
272     // racine de la somme  
273     run("Square Root");  
274     normGrad = getImageID();  
275  
276     return normGrad;  
277 }
```

En prenant en compte le SVH dans le calcul du tatouage par étalement de spectre, nous obtenons le type de signal de tatouage suivant :



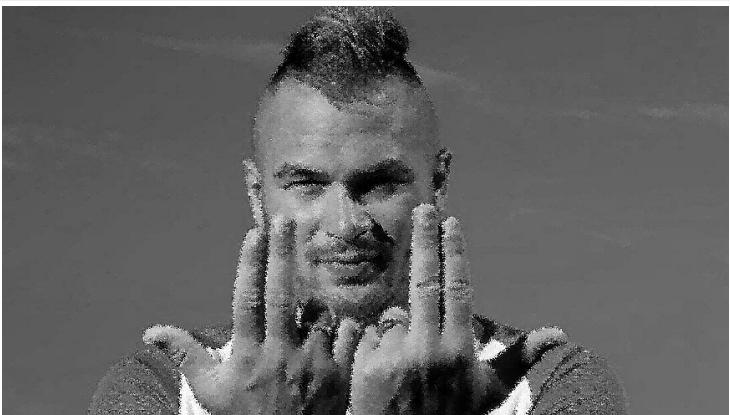
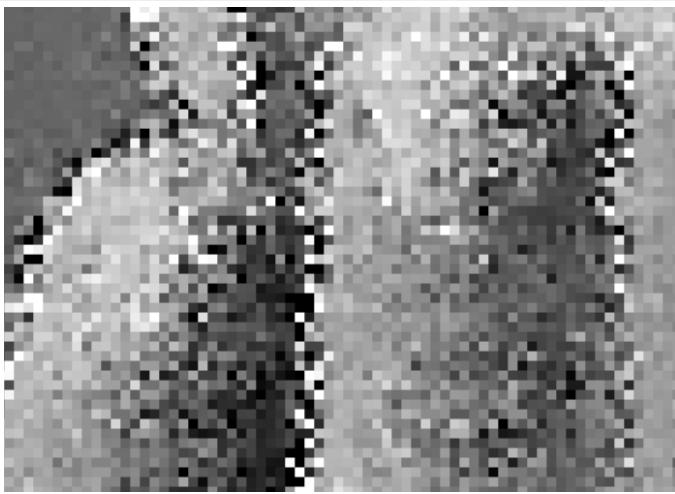
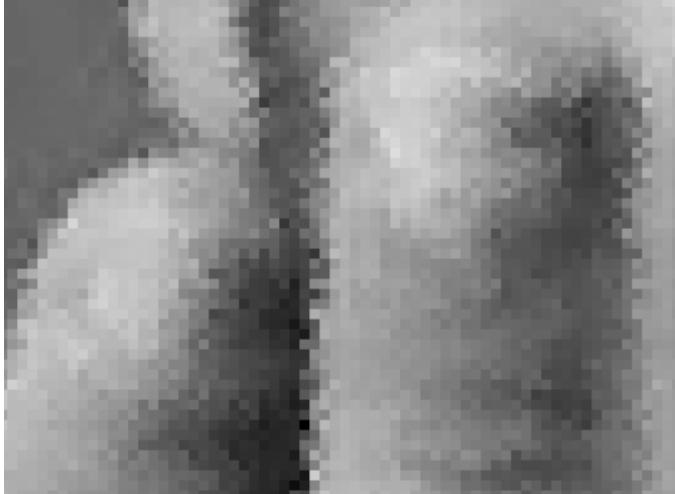
IMAGE – Signal de tatouage de « JUL » avec norme du gradient de Sobel

On observe que le signal est bien moins fort dans les zones de basses fréquences et est, au contraire, plus présent dans les zones de hautes fréquences (ex : contours). Cela est encore plus visible lorsque l'on compare avec les zones de contours de « JUL » affichées grâce à un seuillage de la norme du gradient de Sobel :

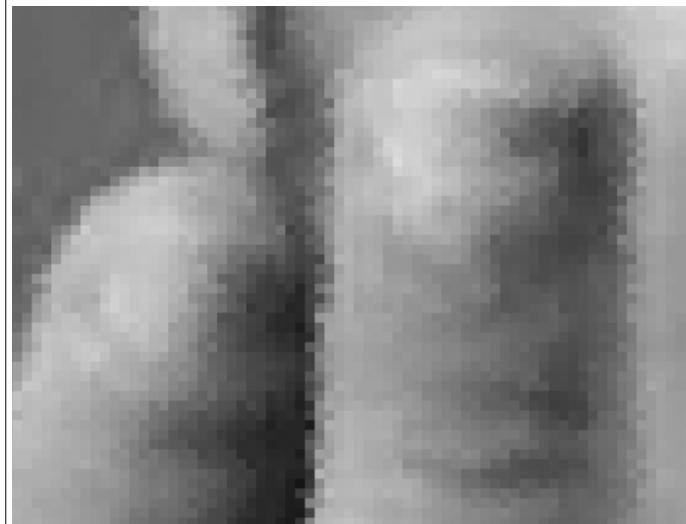


IMAGE – Norme du gradient de Sobel de « JUL » seuillée

Les images tatouées pour les trois PSNR cibles définis sont les suivantes :

PSNR cible	Image tatouée obtenue	Zoom sur une zone de contours
$PSNR_1$		
$PSNR_2$		

$PSNR_3$



En se référant à l'image, on confirme bien que le signal est plus fort dans les hautes fréquences de « *JUL* ».

De plus, à l'instar de la version sans prise en compte du SVH, le PSNR cible permet de réduire la distorsion des hautes fréquences pendant l'insertion du signal de tatouage.

On observe aussi que le PSNR cible pour lequel le signal de tatouage est imperceptible est $PSNR_2=35\text{ dB}$. On en déduit que, grâce à l'exploitation des hautes fréquences d'une image et à la réduction de l'affichage du signal dans les basses fréquences, il est plus facile de dissimuler le signal, ce qui permet de réduire le PSNR cible tout en conservant un signal imperceptible.

Robustesse du schéma de tatouage

Dans cette partie, nous testerons la robustesse du schéma de tatouage implémenté face à deux transformations : l'ajout d'un bruit gaussien et la rotation.

Pour chaque transformation, nous testerons ses effets sur l'image transformée « *JUL PSNR 3* » calculée auparavant, qui est l'image tatouée en utilisant un PSNR cible de 40 dB et en prenant en compte le SVH pendant l'insertion du signal de tatouage.



IMAGE - « JUL PSNR 3 »

Ajout d'un bruit gaussien

Nous commencerons par tester la robustesse de l'image tatouée « *JUL PSNR 3* » transformée par ajout de bruits gaussiens d'écart-types différents. Les expériences menées ici consisteront à prendre l'image transformée « *JUL PSNR 3* », ajouter le bruit gaussien puis décoder le message codé dans l'image tatouée afin de calculer le BER, donnant un indice de robustesse pour le tatouage.

Le tableau ci-dessous montre les expériences réalisées :

Bruit gaussien utilisé	Image tatouée bruitée	BER obtenu
$\sigma=90$		0

$\sigma=100$



0.0625

(1 bit
erroné
sur 16)

$\sigma=120$



0.1875

(3 bits
erronés
sur 16)

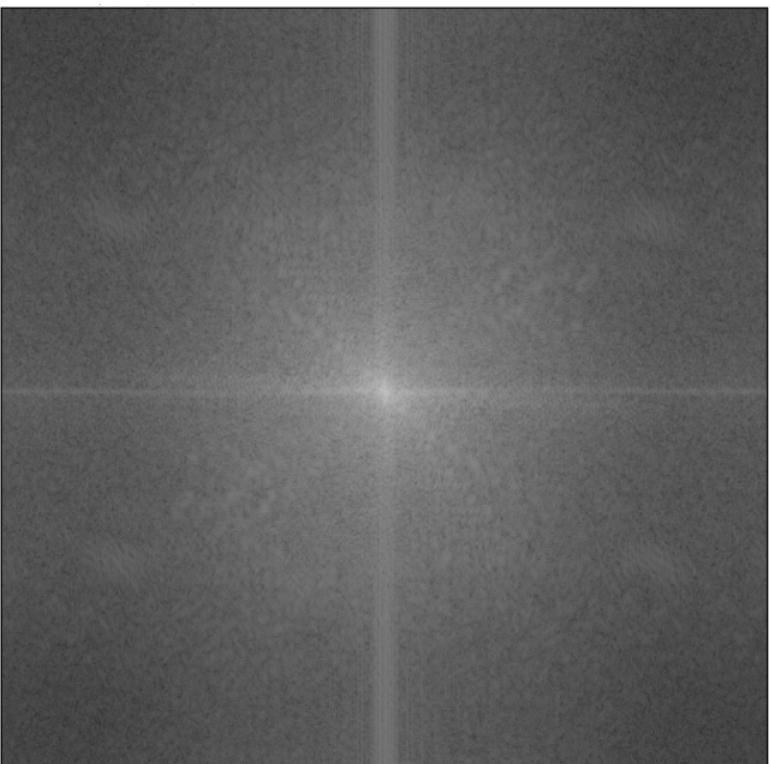
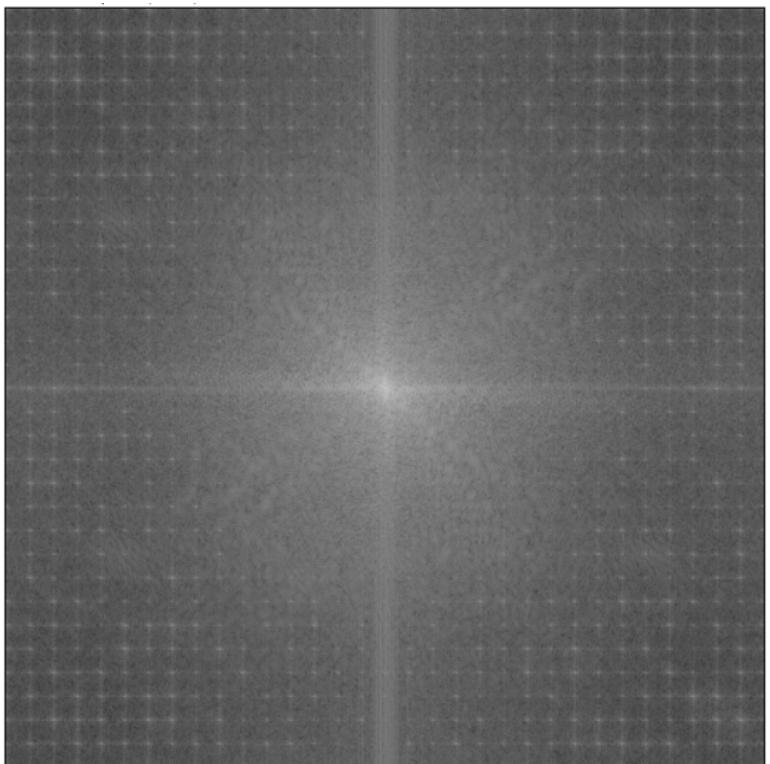
Au vu du bruit trouvé sur l'image et du BER réduit, on remarque que le tatouage est robuste à l'ajout de bruit.

Une explication plausible à cette résistance au bruit est le fait que, lors du tatouage de l'image hôte, chaque tuile (donc chaque porteuse) contient l'entièreté du message. De ce fait, lors du décodage de l'image tatouée transformée, même si un bit du message contenu dans une tuile est corrompu, les autres bits des autres tuiles qui n'ont pas été transformées vont permettre de stabiliser le décodage final. Cela augmente fortement la robustesse du schéma de tatouage. C'est pour cela qu'avec ce système, pour obtenir un bit **i** du message erroné, il faut que la majorité des tuiles possèdent une erreur pour ce bit **i**, ce qui est moins susceptible d'arriver.

Robustesse face à la rotation

Maintenant, nous allons tester la robustesse de l'image « *JUL PSNR 3* » face à la rotation.

Dans un premier temps, nous allons appliquer et comparer les transformées de Fourier (FFT) de l'image hôte (à savoir, « *JUL* ») et de l'image transformée.

FFT de « <i>JUL</i> », l'image hôte	FFT de « <i>JUL PSNR 3</i> », l'image tatouée
 The image shows a dark gray square with a faint, blurry horizontal band running across it, representing the Fourier Transform of the host image 'JUL'.	 The image shows a dark gray square with a distinct grid of white points scattered across it, representing the Fourier Transform of the watermarked image 'JUL PSNR 3'.

En comparant les deux FFT, on remarque que, sur la FFT de l'image tatouée, on retrouve des points blancs qui correspondent aux fréquences des pixels additionnés aux pixels du signal de tatouage. Ceci peut être problématique. En effet, en calculant la FFT, on remarque qu'il est facile de

déetecter qu'une image a été tatouée par étalement de spectre en voyant ces points blancs.

Nous allons maintenant tester la rotation en elle-même en effectuant une rotation de l'image tatouée « *JUL PSNR 3* » de 10°.



IMAGE – « *JUL PSNR 3* » avec rotation de 10°

Le décodage de « *JUL PSNR 3* » avec rotation de 10° donne un **BER = 0.5**. On note donc que le taux d'erreur est trop élevé pour être satisfaisant.

On peut observer l'effet de la rotation sur la FFT de l'image « *JUL PSNR 3* » avec rotation de 10° :

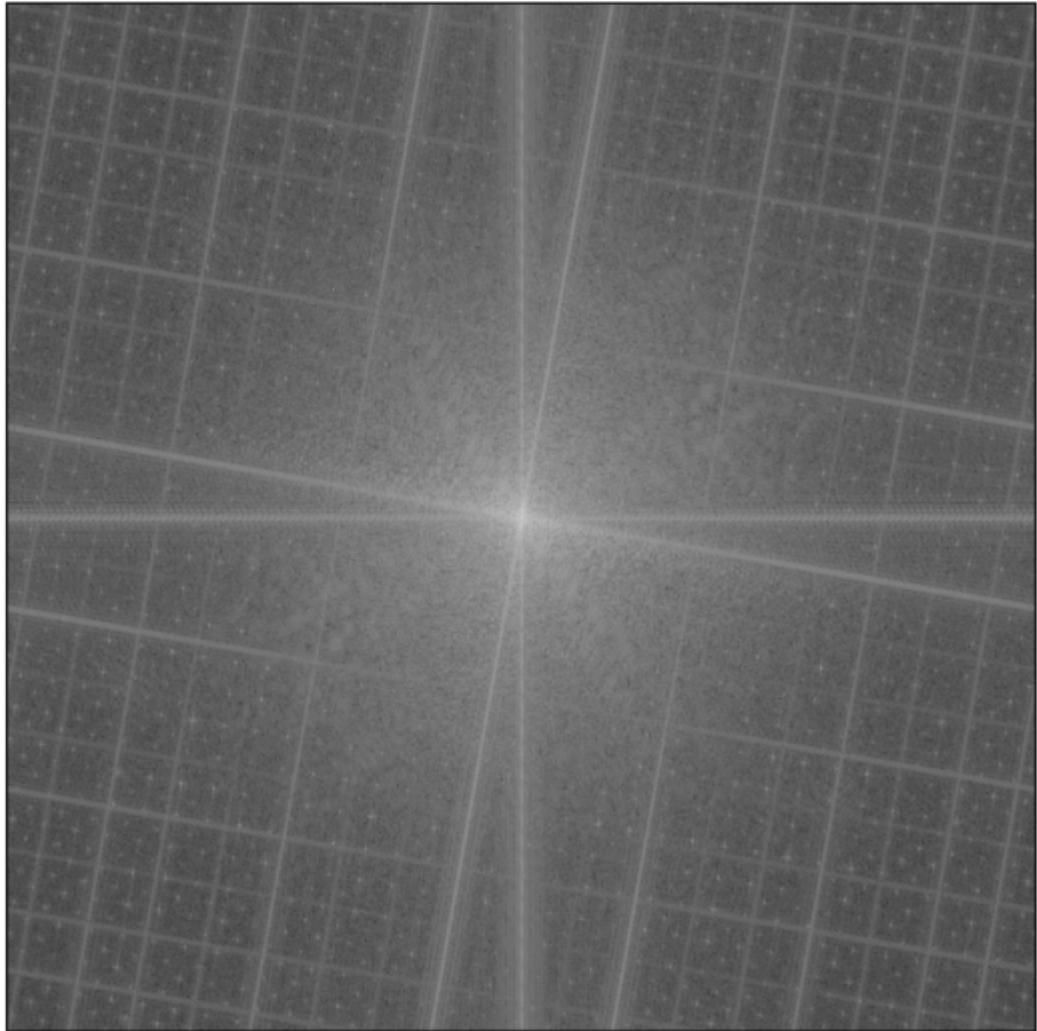


IMAGE – FFT de « JUL PSNR 3 » avec rotation de 10°

On observe ici que les fréquences des points associés aux pixels du signal de tatouage ont également subi la rotation. De ce fait, il est normal que le décodage pose problème étant donné que celui-ci se base sur une estimation et une corrélation des tuiles dont le contenu ne se trouve plus au même endroit.

Une manière de contrer la rotation pour le tatouage serait de calculer la FFT de l'image tatouée qui a subit la rotation, appliquer la rotation inverse sur cette FFT (à savoir ici, -10°), calculer la FFT inverse (FFT^{-1}) puis décoder l'image.