UE 342 : Reconnaissance des formes par corrélation d'images

ZHOU SHUANG
XU Kaiyuan

17 mars 2023

Table des matières

0 Cahier des charges

Le projet consiste à écrire un programme capable de reconnaître et localiser des formes prédéfinies dans une image bitmap. Les différentes taches du programme seront :

- Importer les fichiers bitmap contenant les formes prédéfinis et l'image à analyser.
- Analyser les images par la méthode de corrélation. Pour simplifier les calculs et optimiser les performances les calculs se feront dans l'espace de Fourier, utilisant la transformée de Fourier rapide (FFT) à deux dimensions.

L'analyse des données obtenues dépendra de l'application (soit choisie parmi celles suggérées par l'enseignant soit une application originale que vous proposerez).

La présentation des résultats peut être réalisée soit par la sortie « console » (programme interactif) soit dans un fichier texte, en fonction du problème traité.

1 Introduction

Dans ce projet on code dans Visual Studio Code, en utilisant les outiles mathémathiques, surtout la Transformation de Fourier rapide et des propriétés de la Transformation de Fourier, on va jouer avec des structures complexes pour reconnaissance des formes par corrélation d'image. Pour prévoir, on doit avoir quelquels pixels blanc pour l'image en sortie, où la forme de motif situe, si l'on a bien fait la corrélation.

2 Outils mathématiques

2.1 Nombre complexe

Forme algébrique

$$\underline{z} = \mathfrak{Re}\left(\underline{z}\right) + \imath\mathfrak{Im}\left(\underline{z}\right)$$

Dans l'ensemble des nombres complexes, on définit une multiplication sous la forme algébrique de la manière suivante :

Supposons

$$\begin{aligned}\underline{Z_1} &= \mathfrak{Re}\left(\underline{Z_1}\right) + \imath\mathfrak{Im}\left(\underline{Z_1}\right)\\ \underline{Z_2} &= \mathfrak{Re}\left(\underline{Z_2}\right) + \imath\mathfrak{Im}\left(\underline{Z_2}\right)\end{aligned}$$

Alors

$$\begin{aligned}\underline{Z_1}\cdot\underline{Z_2} &= \left(\mathfrak{Re}\left(\underline{Z_1}\right) + \imath\mathfrak{Im}\left(\underline{Z_1}\right)\right)\cdot\left(\mathfrak{Re}\left(\underline{Z_2}\right) + \imath\mathfrak{Im}\left(\underline{Z_2}\right)\right)\\ &= \left(\mathfrak{Re}\left(\underline{Z_1}\right)\mathfrak{Re}\left(\underline{Z_2}\right) - \mathfrak{Im}\left(\underline{Z_1}\right)\mathfrak{Im}\left(\underline{Z_2}\right)\right) + \imath\left(\mathfrak{Re}\left(\underline{Z_1}\right)\mathfrak{Im}\left(\underline{Z_2}\right) + \mathfrak{Re}\left(\underline{Z_2}\right)\mathfrak{Im}\left(\underline{Z_1}\right)\right)\end{aligned}$$

Forme polaire

$$\underline{Z} = |\underline{Z}|\ \mathrm{e}^{\imath\arg(\underline{Z})}$$

où la norme :

$$|\underline{Z}| = \sqrt{\left(\mathfrak{Re}\left(\underline{Z}\right)\right)^2 + \left(\mathfrak{Im}\left(\underline{Z}\right)\right)^2}$$

l'argument :

$$\arg(\underline{Z}) = \begin{cases}\arctan\left(\frac{\mathfrak{Im}(\underline{Z})}{\mathfrak{Re}(\underline{Z})}\right) & \text{si } \mathfrak{Re}\left(\underline{Z}\right) > 0\\ \arctan\left(\frac{\mathfrak{Im}(\underline{Z})}{\mathfrak{Re}(\underline{Z})}\right) + \pi & \text{si } \mathfrak{Re}\left(\underline{Z}\right) < 0\\ \mathrm{sgn}(\mathfrak{Im}\left(\underline{Z}\right))\frac{\pi}{2} & \text{si } \mathfrak{Re}\left(\underline{Z}\right) = 0\end{cases}$$

Racine n-ième de l'unité
Pour un entier naturel non nul N donné, on appelle racine n-ième de l'unité toute solution complexe de l'équation :

$$\underline{Z}^N = 1$$

d'inconnue $\underline{Z}$. Il existe exactement n racines n-ièmes de l'unité.

Pour un entier N donné, les racines n-ièmes de l'unité sont situées sur le cercle unité du plan complexe et sont les sommets d'un polygone régulier à N côtés. Donc l'angle entre les racines i-ièmes est (i+1)-ièmes vaut $\frac{2\pi}{N}$.

Les racines n-ièmes de l'unité peuvent donc s'écrire sous la forme :

$$\boxed{W_N^k = \mathrm{e}^{\imath 2\pi k/N} \quad (N \in \mathbb{N}^* \text{ et } k \in \{0, 1, \ldots, n-1\})} \tag{2.1.1}$$

Dans la section suivante, on va simplier l'écriture de la matrice DFT (**discrete Fourier transform**) en utilisant le cas $k = 1$ ici, on a

$$W_N^1 = \mathrm{e}^{\imath 2\pi 1/N} \quad (N \in \mathbb{N}^* \text{ et } k = 1)$$

soit

$$W_N = \mathrm{e}^{\imath 2\pi/N} \quad (N \in \mathbb{N}^*) \tag{2.1.2}$$

De plus, pour un entier naturel non nul pair N donné, on va noter deux propretés de la racine n-ième de l'unité :

$$W_N^{k+N} = W_N^k \quad \text{(périodicité)} \tag{2.1.3}$$

$$W_N^{k+\frac{N}{2}} = -W_N^k \quad \text{(symétrie)} \tag{2.1.4}$$

Car

$$W_N^{k+N} = \mathrm{e}^{\imath 2\pi(k+N)/N} = \mathrm{e}^{\imath 2\pi(k)/N} \cdot \mathrm{e}^{\imath 2\pi(N)/N} \cdot 1 = \mathrm{e}^{\imath 2\pi(k)/N} = W_N^k$$

et

$$W_N^{k+N} = \mathrm{e}^{\imath 2\pi(k+\frac{N}{2})/N} = \mathrm{e}^{\imath 2\pi(k)/N} \cdot \mathrm{e}^{\imath 2\pi(\frac{N}{2})/N} = \mathrm{e}^{\imath 2\pi(k)/N} \cdot -1 = -W_N^k$$

2.2 Transformation de Fourier

Définition Soit $f \in L^1(\mathbb{R})$, sa transformée de Fourier s'écrit

$$\boxed{F(\nu) = \int_{-\infty}^{+\infty} f(x)\ \mathrm{e}^{-\imath 2\pi\nu x}\ \mathrm{d}x} \tag{2.2.1}$$

où ν est la fréquence en $s^{-1} = Hz$; on a la pulsation correspondante $\omega = 2\pi\nu$ en $rad.s^{-1}$.

Inversion : si $f \in L^1(\mathbb{R})$ et $F \in L^1(\mathbb{R})$, alors sa transformée de Fourier inverse s'écrit

$$f(x) = \int_{-\infty}^{+\infty} F(\nu)\ \mathrm{e}^{+\imath 2\pi\nu x}\ \mathrm{d}\nu \tag{2.2.2}$$

Propété

$$f(x-a)\ \underrightarrow{TF}\ \mathrm{e}^{-\imath 2\pi\nu a}\ F(\nu) \quad \text{(théorème de retard)} \tag{2.2.3}$$

Preuve on pose $y = x - a$ donc $\mathrm{d}y = \mathrm{d}x$

$$\begin{aligned}\int_{-\infty}^{+\infty} f(x-a)\ \mathrm{e}^{-\imath 2\pi\nu x}\ \mathrm{d}x &= \int_{-\infty}^{+\infty} f(y)\ \mathrm{e}^{-\imath 2\pi\nu(y+a)}\ \mathrm{d}y \\ &= \mathrm{e}^{-\imath 2\pi\nu a} \int_{-\infty}^{+\infty} f(y)\ \mathrm{e}^{-\imath 2\pi\nu(y)}\ \mathrm{d}y \\ &= \mathrm{e}^{-\imath 2\pi\nu a}\ F(\nu)\end{aligned}$$

Propété Soit $f \in L^1(\mathbb{R})$ de classe C^n telle que $f^{(k)} \in L^1(\mathbb{R})$ où $0 \leq k \leq n$, alors :

$$f^{(k)}(x)\ \underrightarrow{TF}\ (\imath 2\pi\nu)^k\ F(\nu) \quad \text{(dérivation temporelle)} \tag{2.2.4}$$

Preuve Comme

$$F(\nu) = \int_{-\infty}^{+\infty} f(x)\ \mathrm{e}^{-\imath 2\pi\nu x}\ \mathrm{d}x$$

alors

$$\begin{aligned}TF\left[f^{(k)}(x)\right] &= \int_{-\infty}^{+\infty} f^{(k)}(x)\ \mathrm{e}^{-\imath 2\pi\nu x}\ \mathrm{d}x \\ &= (-1)^k \int_{-\infty}^{+\infty} f(x)\ (-\imath 2\pi\nu)^k\ \mathrm{e}^{-\imath 2\pi\nu x}\ \mathrm{d}x \\ &= (\imath 2\pi\nu)^k\ F(\nu)\end{aligned}$$

Remarque $(-1)^k$ car

$$f \in L^1(\mathbb{R}) \Rightarrow \lim_{x \to \pm\infty} f(x) = 0$$

$$\int u'v = [uv] - \int uv' = 0 - \int uv' = (-1)^1 \int uv'$$

$$\begin{aligned}\int u''v = [u'v] - \int u'v' &= 0 - \int uv' = (-1)^1 \int u'v' \\ &= -\left([uv'] - \int uv''\right) = -\left(0 - \int uv''\right) = (-1)^2 \int uv''\end{aligned}$$

2.3 Fast Fourier Transformation

2.3.1 Préliminaire

On part à la définition de **Transformation de Fourier** (TF) (2.2.1) :

$$\boxed{F(\nu) = \int_{-\infty}^{+\infty} f(x)\ \mathrm{e}^{-\imath 2\pi\nu x}\ \mathrm{d}x}$$

Lorsque le signal à traiter n'est plus analogique mais numérique, on introduit l'expréssion de la **Transformée de Fourier à temps discret** (TFtd) du signal $\{f_k = f(x_k) = f(kT_e)\}_{k\in\mathbb{Z}}$:

$$\begin{aligned} F(\nu) &= \sum_{k=-\infty}^{+\infty} f\left(x_k\right) e^{-i2\pi\nu(x_k)} \\ &= \sum_{k=-\infty}^{+\infty} f\left(kT_e\right) e^{-i2\pi\nu(kT_e)} \\ &= \sum_{k=-\infty}^{+\infty} f_k\ e^{-i2\pi\nu kT_e} \end{aligned}$$

avec $k \in \mathbb{Z}$,
où $f_k = f(kTe)$: le k-ème échantillon f_k est la valeur du signal $f(x)$ prélevée à l'instant $x_k = kT_e$. Et $F(\nu)$ est appelé TFtd de $f(kT_e)$ ou encore spectre de $f(kT_e)$. La variable ν est continue et représente la fréquence en Hz

Pour un signal de durée N, on définit la **Transformée de Fourier discrete** (TFD) par une approximation de l'intégration dans (2.2.1)

$$\begin{aligned} F\left(\nu_n\right) &\approx \sum_{k=0}^{N-1} f\left(x_k\right) e^{-i2\pi\nu_n x_k}\ T_e \\ &= \sum_{k=0}^{N-1} f\left(kT_e\right) e^{-i2\pi(\frac{n}{N}F_e)kT_e}\ T_e \\ &= T_e \sum_{k=0}^{N-1} f_k\ e^{-i2\pi\frac{n}{N}k} \end{aligned}$$

avec $k \in [0, N-1]$, $n \in \left[-\frac{N}{2}, \frac{N}{2} - 1\right]$
La TFD est la version discrétisée en fréquence de la TFtd : elle est calculée en toutes les fréquences $\nu_n = \frac{n}{N}F_e$ avec $n \in [0, N-1]$ ou bien $n \in I$ avec I n'importe quel ensemble de la forme $[a, a+N-1]$: comme le TF est périodique de période F_e, peu importe l'intervalle du moment qu'il contient N échantillons.

Finalement, on obtient la formule de la Transformée de Fourier discrete :

$$\boxed{F_n \equiv \sum_{k=0}^{N-1} f_k\ e^{-i2\pi\frac{n}{N}k}} \tag{2.3.1}$$

La transformée de Fourier discrète transforme N nombres complexes (les f_k) en N nombres complexes (les F_n). Elle ne dépend d'aucun paramètre dimensionnel, tel que l'échelle de temps T_e. La relation entre la transformée de Fourier discrète d'un ensemble de nombres et leur transformée de Fourier continue lorsqu'ils sont considérés comme des échantillons d'une fonction continue échantillonnée à un intervalle T_e peut être réécrite comme :

$$F\left(\nu_n\right) \approx T_e F_n$$

2.3.2 Notation

Regardons (2.2.1), on note

$$W_N = \mathrm{e}^{\imath 2\pi/N} \qquad (N \in \mathbb{N}^*)$$

On commence par la Transformée de Fourier discrete (2.3.1) :

$$\begin{aligned} F_n &\equiv \sum_{k=0}^{N-1} f_k\ e^{-i2\pi\frac{n}{N}k} \\ &= \sum_{k=0}^{N-1} f_k\ e^{-i2\pi\frac{1}{N}nk} \\ &= \sum_{k=0}^{N-1} f_k\ (W_N)^{-nk} \end{aligned}$$

Soit

$$\boxed{F_n = \sum_{k=0}^{N-1} W_N^{-nk} \ f_k} \tag{2.3.2}$$

On peut aussi écrire (2.3.2) sous forme maticielle :

$$\begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_k \\ \vdots \\ F_{N-1} \end{bmatrix} = \begin{bmatrix} W_N^{-(0)\times(0)} & W_N^{-(0)\times(1)} & \dots & W_N^{-(0)\times(N-1)} \\ W_N^{-(1)\times(0)} & W_N^{-(1)\times(1)} & \dots & W_N^{-(1)\times(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ W_N^{-(k)\times(0)} & W_N^{-(k)\times(1)} & \dots & W_N^{-(k)\times(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ W_N^{-(N-1)\times(0)} & W_N^{-(N-1)\times(1)} & \dots & W_N^{-(N-1)\times(N-1)} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_k \\ \vdots \\ f_{N-1} \end{bmatrix} \tag{2.3.3}$$

Maintenant on chercher le k-ième de composant en regardant (2.3.1) :

$$F_k = \sum_{j=0}^{N-1} e^{-i2\pi \frac{j}{N} k} \ f_j \tag{2.3.4}$$

$$F_k = \sum_{j=0}^{\frac{N}{2}-1} e^{-i2\pi \frac{(2j)}{N} k} \ f_{2j} + \sum_{j=0}^{\frac{N}{2}-1} e^{-i2\pi \frac{(2j+1)}{N} k} \ f_{2j+1} \tag{2.3.5}$$

$$F_k = \sum_{j=0}^{\frac{N}{2}-1} e^{-i2\pi \frac{(j)}{\frac{N}{2}} k} \ f_{2j} + W_N^{-k} \sum_{j=0}^{\frac{N}{2}-1} e^{-i2\pi \frac{(j)}{\frac{N}{2}} k} \ f_{2j+1} \tag{2.3.6}$$

$$F_k = F_k^{even} + W_N^{-k} \ F_k^{odd} \tag{2.3.7}$$

2.3.3 Cooley–Tukey FFT algorithme

Dans la partie suivante, on suppose $N = 2^m, m \in \mathbb{Z}$!

Méthode Quand on fait cette méthode (Cooley–Tukey FFT algorithm) parlée en haut on doit avoir

$$\begin{cases} F_k = F_k^{even} + W_N^{-k} \ F_k^{odd} \\ F_{k+\frac{N}{2}} = F_k^{even} + W_N^{-(k+\frac{N}{2})} \ F_k^{odd} \end{cases}$$

en utillisant la propriété de la symérie de la Racine n-ième de l'unité (2.1.4)

$$W_N^{k+\frac{N}{2}} = -W_N^k \quad \text{(symétrie)}$$

on trouve

$$\begin{cases} F_k = F_k^{even} + W_N^{-k} \ F_k^{odd} \\ F_{k+\frac{N}{2}} = F_k^{even} - W_N^{-k} \ F_k^{odd} \end{cases} \tag{2.3.8}$$

C'est magique que une fois on sait F_k^{even}, F_k^{odd} (Attension ici element de vecteur taille $\frac{N}{2} \times 1$), on calcule W_N^{-k} (Attension ici element de vecteur taille $\frac{N}{2} \times \frac{N}{2}$, mais divisé par N en exponentielle), on obtient F_k et $F_{k+\frac{N}{2}}$ en même temps.

Chaque fois on réduit la taille de vecteur colonne F_k par 2, comme on suppse $N = 2^m, m \in \mathbb{Z}$, il existe une transformation en un point qui n'est qu'un des nombres d'entrée f_n :

$$F_k^{eoeeoeo\cdots oee} = f_n \tag{2.3.9}$$

(Bien entendu, cette transformation en un point ne dépend pas de k, puisqu'elle est périodique en k avec une période de 1).

Complexité en temps Dans cette méthode le complexité en temps vaut $O(Nlog_2N)$, car pour une list de f_i dans $\{f_0, f_1, ..., f_{N-1}\}$ où $N = 2^m, m \in \mathbb{Z}$ donné

On consière la méthode de Dichotomie, d'abord on fait un diviseur en deux, on utilise la méthode Dichotomie à savoir si l'élement f_i est dans cette list, donc on a fait une comparaison, on peut noter f_N comme :

$$f(N) = 1 + f(\frac{N}{2^1})$$

on fait la 2-ième fois :

$$f(N) = 1 + f(\frac{N}{2}) \tag{2.3.10}$$
$$= 1 + (1 + f(\frac{N}{4})) \tag{2.3.11}$$
$$= 2 + \frac{N}{2^2} \tag{2.3.12}$$

donc on sait que si on fait ça m-ième fois, on ne peut plus diviser en deux, on trouve 1 seul élement :

$$f(N) = m + f(\frac{N}{2^m}) \tag{2.3.13}$$
$$= m + f(1) = m + 1 \tag{2.3.14}$$

soit

$$\frac{N}{2^m} = 1$$

soit

$$N = 2^m$$

soit

$$m = log_2 N$$

Comme on doit faire m-ième fois pour obtenir le résultat, donc ici pour la méthode Dichotomi on a le Complexité en temps vaut $O(Nlog_2 N)$;

De plus on a N éléments en totel, comme le Complexité en temps pour chaque élélment vaut $O(log_2 N)$, le Complexité pour Cooley–Tukey FFT algorithme vaut $O(N \times log_2 N)$.

Mais comme si on le calcule en utilisant la Transformée de Fourier discrete (2.3.2) directement, le Complexité vaut $O(N \times N)$, car il y a N élément est chaque élément doit faire N fois la multuiplication. Ici, on conclure que la méthode de Cooley–Tukey FFT algorithme fait un calcul plus rapide.

2.3.4 Bit reversal

On revient à les data $\{f_0, f_1, ..., f_{N-1}\}$ où $N = 2^m, m \in \mathbb{Z}$, comment on doit envoyer la list soit le vecteur de f_k pour à la sortie on peux bien trouver F_k à l'order, pour ça on réfléchit inversement, on prend ici $N = 8$ par exemple :

$$f_0 \quad f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7$$

$$f_0^e \quad f_2^e \quad f_4^e \quad f_6^e \mid f_1^o \quad f_3^o \quad f_5^o \quad f_7^o$$

$$f_0^{ee} \quad f_4^{ee} \mid f_2^{eo} \quad f_6^{eo} \mid f_1^{oe} \quad f_5^{oe} \mid f_3^{oo} \quad f_7^{oo}$$

$$f_0^{eee} \mid f_4^{eeo} \mid f_2^{eoe} \mid f_6^{eoo} \mid f_1^{oee} \mid f_5^{oeo} \mid f_3^{ooe} \mid f_7^{ooo}$$

On pose even $= 1$ et odd $= 0$:

$$f_0^{000} \mid f_4^{001} \mid f_2^{010} \mid f_6^{011} \mid f_1^{100} \mid f_5^{100} \mid f_3^{101} \mid f_7^{111}$$

Bit reversal $\leftarrow$

$$f_0^{000} \mid f_4^{100} \mid f_2^{010} \mid f_6^{110} \mid f_1^{001} \mid f_5^{101} \mid f_3^{101} \mid f_7^{011}$$

On bien vu que après avoir fait Bit reversal, on trouve l'indice en haut de f_k en base 2 vaut exactement l'indice en basse en base 10, par exemple pour f_6^{110}, $(110)_2 = (6)_{10}$, donc au début on doit placer en ordre $\{f_0, f_4, f_2, f_6, f_1, f_5, f_3, f_7\}$ et on a une application avec $\{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$.

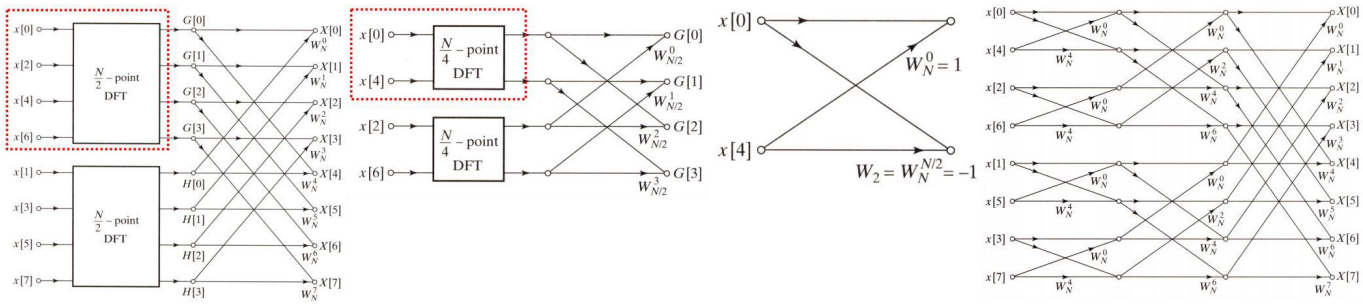

FIGURE 2.3.1 – Data flow diagramme pour $N = 8$ (`https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm#/media/File:DIT-FFT-butterfly.svg`)

2.3.5 FFT inverse

Je laisse une petite section à parler de FFT inverse : quand on fait l'inverse de matrice de TFD ou nommé Matrice de Vandermonde-Fourier, il est suffit de changer une ligne de code à la fin, est les autres reste comme FFT. Soit on remplce $\omega = e^{\frac{i2\pi}{n}}$ dans le code de FFT par $\frac{1}{n}\omega^{-1}$ pour calculer FFT inverse.

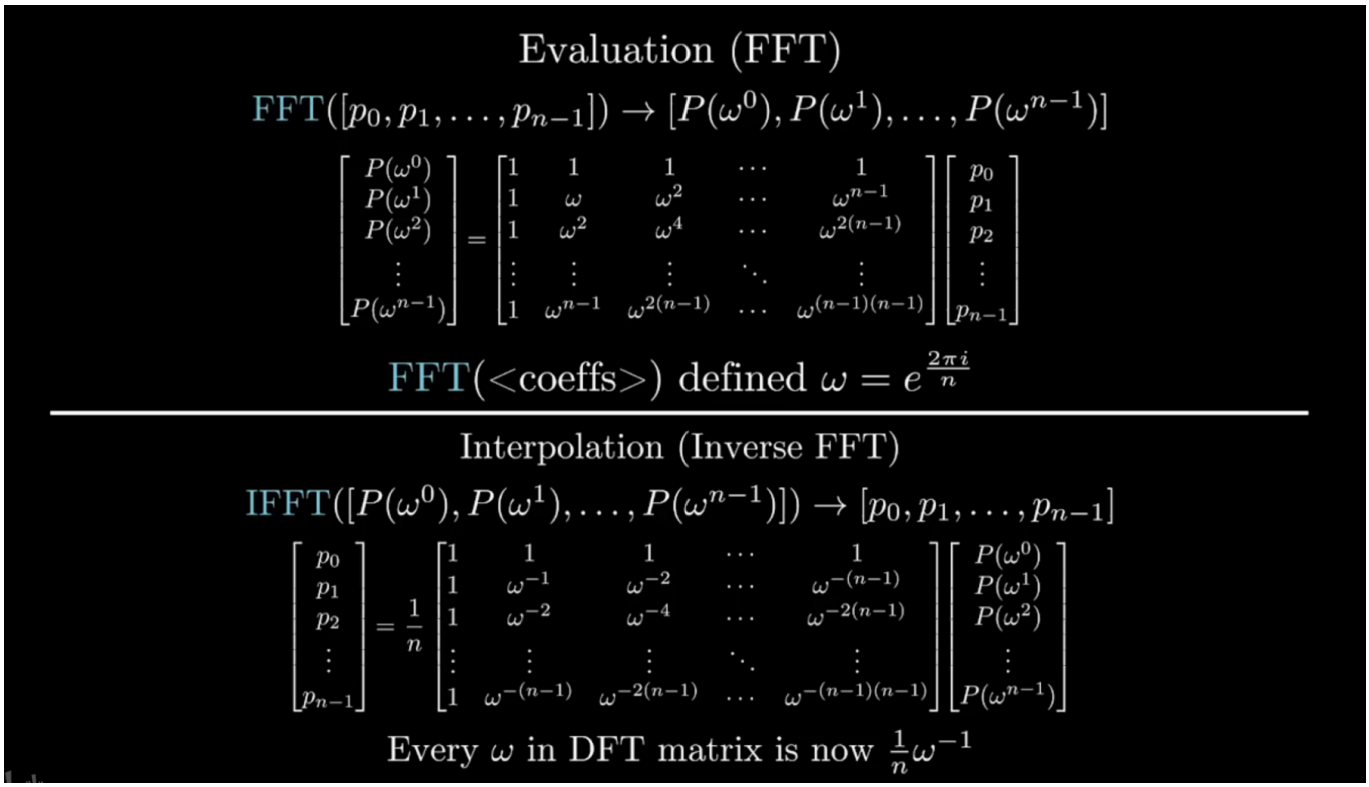

FIGURE 2.3.2 – FFT inverse (`https://www.youtube.com/watch?v=h7apO7q16V0&feature=emb_logo`)

2.4 Théorème de la Correlation

La correlation entre deux fonctions, nomée $\text{Corr}(g, h)$, est donnée par

$$\text{Corr}(g, h) \equiv \int_{-\infty}^{+\infty} g(x + t)\ h(t)\ \mathrm{d}t \tag{2.4.1}$$

La corrélation est une fonction de x (soit en temporelle), que l'on appelle le *lag*. Elle se situe donc dans le domaine temporel et s'avère être un membre de la paire de transformées :

$$\boxed{\text{Corr}(g, h) \ \underset{\rightarrow}{TF}\ G(\nu)H^*(\nu) \quad \text{(théorème de la correlation)}} \tag{2.4.2}$$

2.5 Normalisation : normalisation et conversion à entier

Ici on a une suite de data assezc grande $\{I_{min}, ..., I_i, ..., I_{max}\}$, on veut faire une normalision linéaire pour que I_{min} lie à 0 et I_{max} lie à 255 à partir de I_i connue :

$$\frac{I_{max} - I_{min}}{255 - 0} = \frac{I_i - I_{min}}{\text{entier à chercher} - 0}$$

donc

$$\boxed{\text{entier à chercher} = \text{round}\left(\frac{I_i - I_{min}}{I_{max} - I_{min}} \times 255\right)} \tag{2.5.1}$$

2.6 Filtre de Gauss

Le filtre de Gauss est, en électronique et en traitement du signal, un filtre dont la réponse impulsionnelle est une fonction gaussienne. Le filtre de Gauss minimise les temps de montée et de descente, tout en assurant l'absence de dépassement en réponse à un échelon. Cette propriété est étroitement liée au fait que le filtre de Gauss présente un retard de groupe minimal. (`https://en.wikipedia.org/wiki/Gaussian_filter`)
Dans $1D$, on a :

$$g(x) = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot e^{-\frac{x^2}{2\sigma^2}}$$

Comme dans notre cas pratrique, on va traiter les images, donc c'est le cas en $2D$:

$$\boxed{g(x, y) = \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}} \tag{2.6.1}$$

3 Manipulation

3.0 plan global

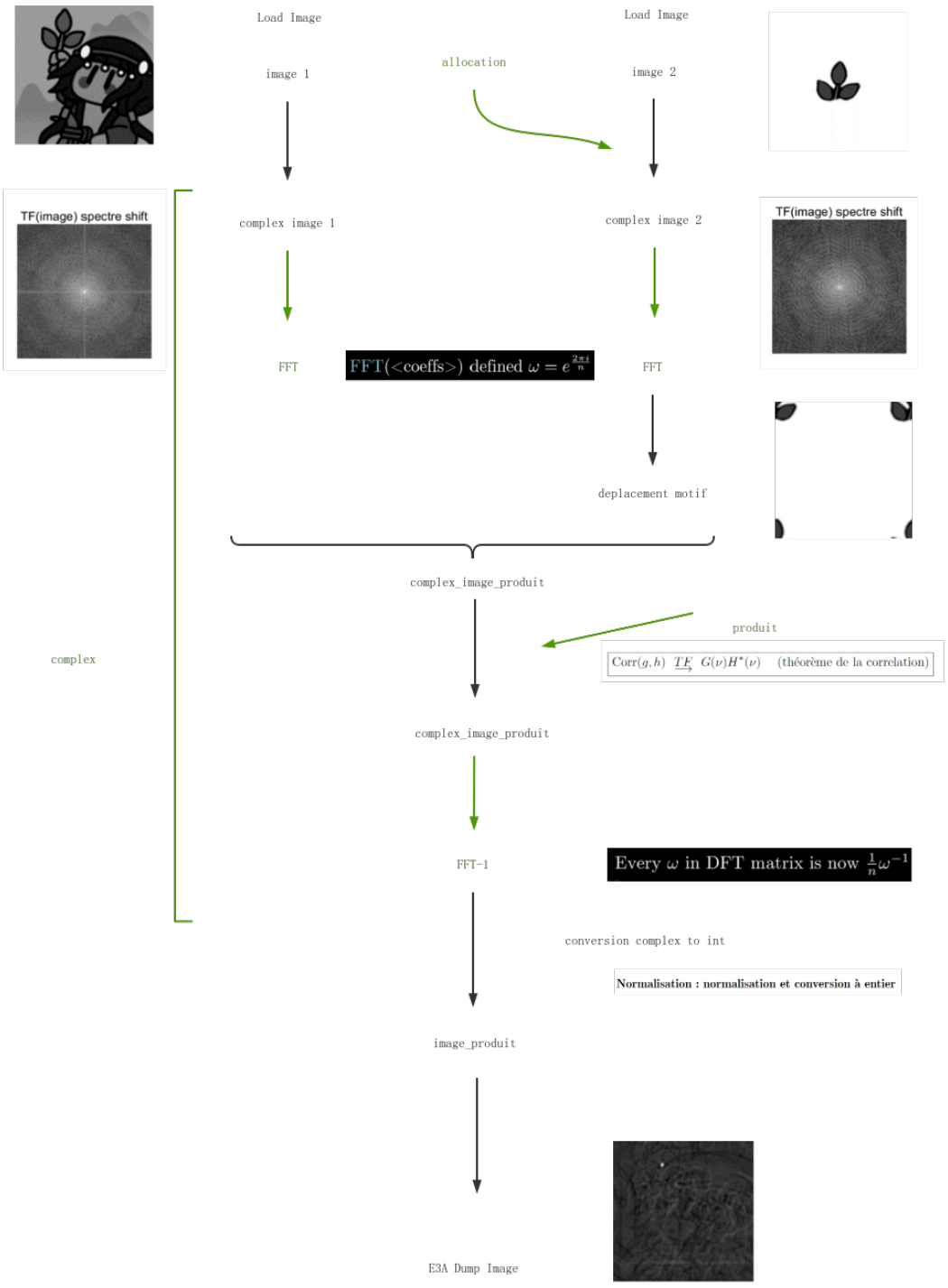

FIGURE 3.0.1 – Plan global

Listing 1 – code mian.c à commenter

```c
#include <stdio.h>
#include <stdlib.h>
#include "projet.h"
#define case 55

int main(){
//---------------------image1 : image a tester ----------------------//
bwimage_t image1;
error_e retval=E3A_OK;
retval=E3ALoadImage("../../image/main_tester_input.png", &image1);

Imagecomplexe complex_image1 = conversion_image2complexe(image1);

fft2(complex_image1, 1); // fft

//---------------------image2 : motif centre-------------------------//
bwimage_t image2;

retval=E3ALoadImage("../../image/main_motif_input.png", &image2);

Imagecomplexe complex_image2 = conversion_image2complexe(image2);

fft2(complex_image2, 1); // fft

//------------motif_deplace :  motif deplace au 4 coins ------------//
Imagecomplexe motif_deplace = deplacer_motif(complex_image2);

//------image produit image :  entre image1 et motif_deplace -------//
Imagecomplexe complex_image_produit_image=produit_image_version2(complex_image1, motif_deplace)
    ; // produit_image_version2 : meme principe que test_corr2.c ou il n'y a pas de terme de
    derivee croise

//--------------------fft inverse---------------------//
fft2(complex_image_produit_image, -1);

//----------------------conversion complexe2image-------------------------//
bwimage_t  int_image = conversion_complexe2image(complex_image_produit_image);


//--------------------DumpImage----------------------//
E3ADumpImage("../../image/main_tester_output.png", &int_image);


release_tab_c(complex_image_produit_image.crawdata); //release_tab_c : c'est le tab complexe
    utilise dans la structure nomee complex_image_produit_image

return 0;

}
```

Avec image 1 et image 2 utilisées dans ce code main.c :

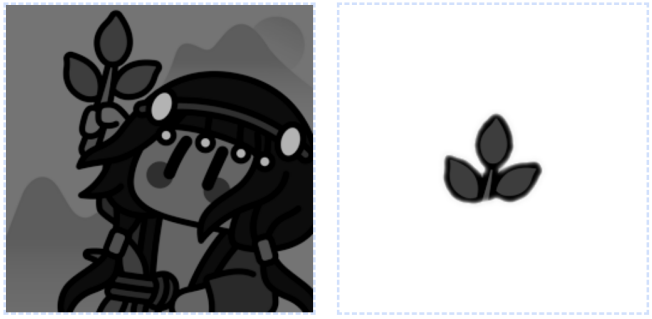

FIGURE 3.0.2 – image 1 et image 2 utilisées dans ce code main.c

image 1 est l'image à tester ou dit image de référence ; image 2 est le motif.

3.1 configuration environnement

Comme c'est notre première d'utiliser le Visual Studio Code à faire le traitement d'image, j'ai taillé ici comment on fait la configuration à partir de zéro :

Step 1 : Créer un nouveau dossier

FIGURE 3.1.1 – Step 1 : Créer un nouveau dossier

Step 2 : Ouvrir dans Visual Studio Code

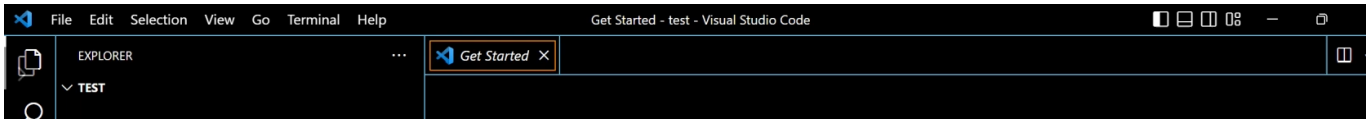

FIGURE 3.1.2 – Step 2 : Ouvrir dans Visual Studio Code

Step 3 : Cliquer sur View plus cliquer sur Command Palette

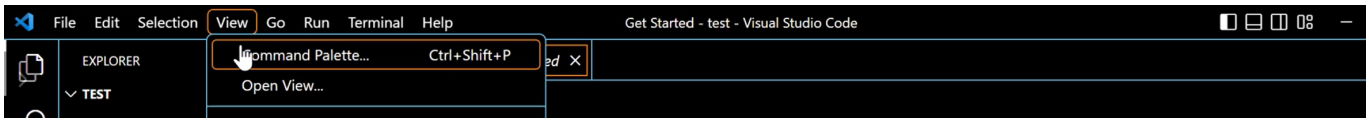

FIGURE 3.1.3 – Step 3 : Cliquer sur View plus cliquer sur Command Palette

Step 4 : Cliquer sur Cmake Configure

FIGURE 3.1.4 – Step 4 : Cliquer sur Cmake Configure

Step 5 : Cliquer sur Visual Studio Community 2022 Realease - adm64

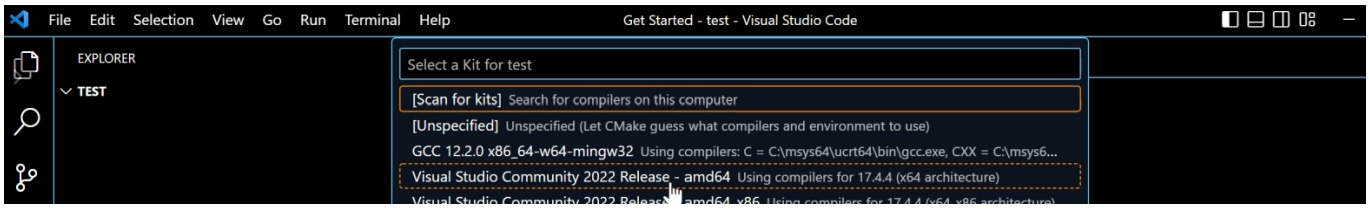

FIGURE 3.1.5 – Step 5 : Cliquer sur Visual Studio Community 2022 Realease - adm64

Step 6 : Cliquer sur OK puis cliquer sur OK aussi

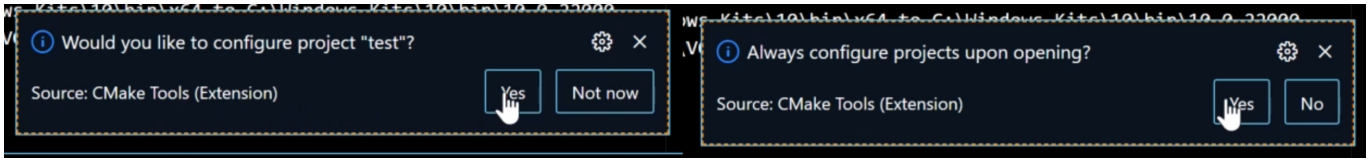

FIGURE 3.1.6 – Step 6 : Cliquer sur OK puis cliquer sur OK aussi

Step 7 : Dans le dossier test, créer le fichier CmakeLists.txt

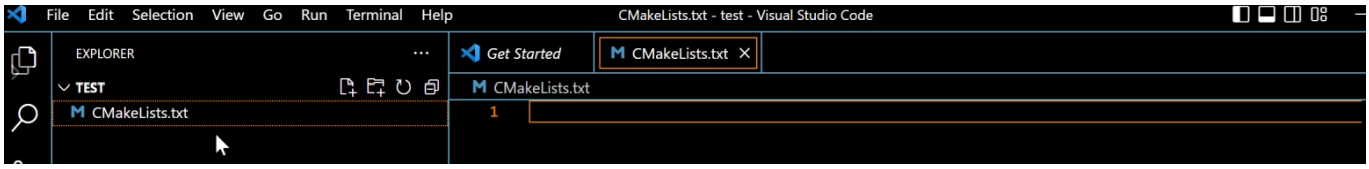

FIGURE 3.1.7 – Step 7 : Dans le dossier test, créer le fichier CmakeLists.txt

Step 8 : Taper le text suivant :

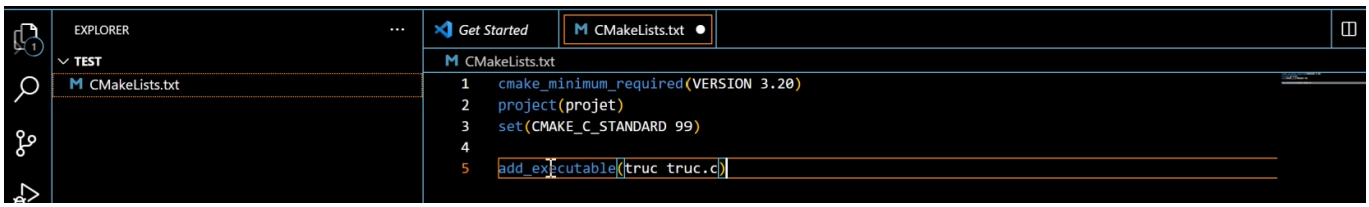

FIGURE 3.1.8 – Step 8 : Taper le text suivant

Step 9 : build

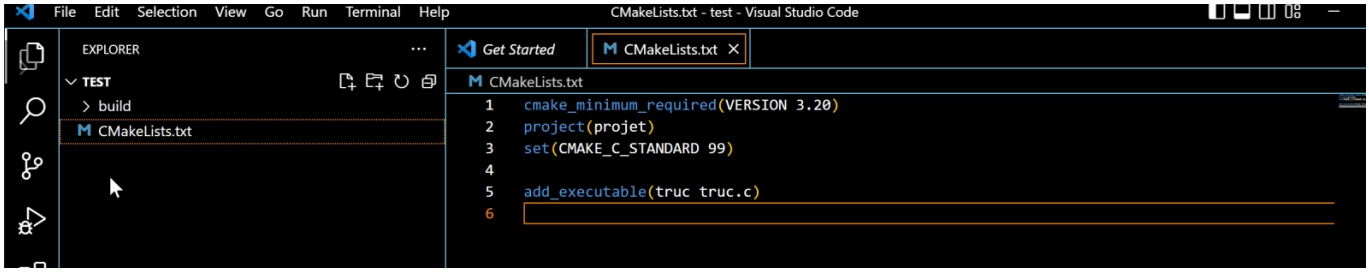

FIGURE 3.1.9 – Step 9 : build

Step 10 : Créer le fichier.c ici truc.c :

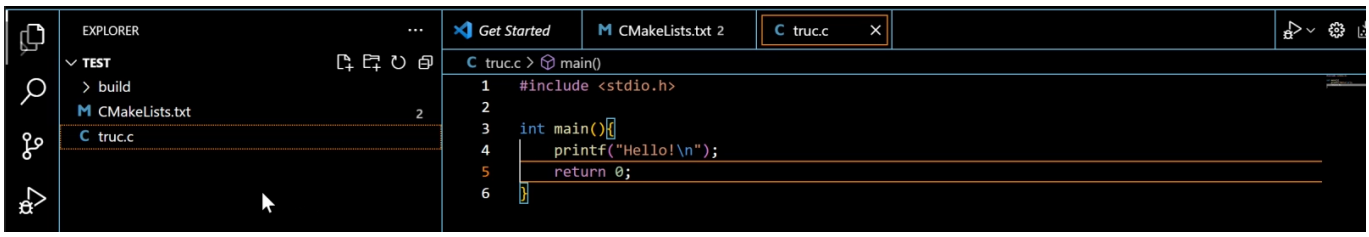

FIGURE 3.1.10 – Step 10 : Créer le fichier.c ici truc.c

Step 11 : Cliquer sue Cmake à gauche, puis cloquer sur ... en haut, choisir Clean Reconfigure All Projects :

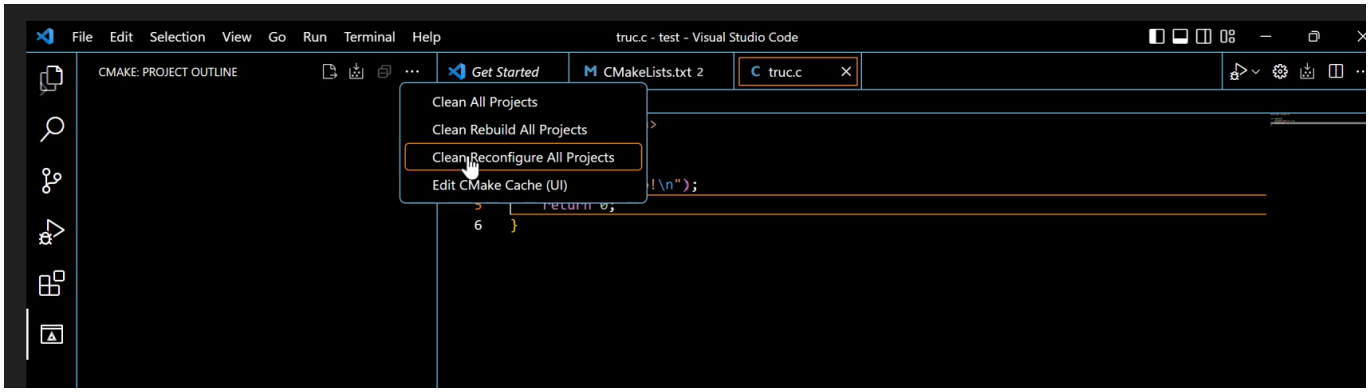

FIGURE 3.1.11 – Step 11 : Cliquer sur Cmake à gauche, puis cloquer sur ... en haut, choisir Clean Reconfigure All Projects

Step 12 : Cliquer sur All Build en basse

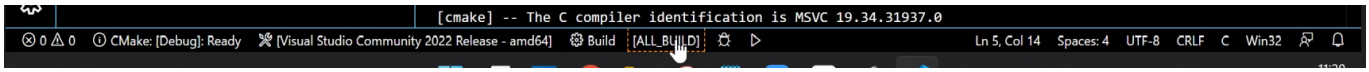

FIGURE 3.1.12 – Step 12 : Cliquer sur All Build en basse

Step 13 : Cliquer sur truc EXECUTABLE

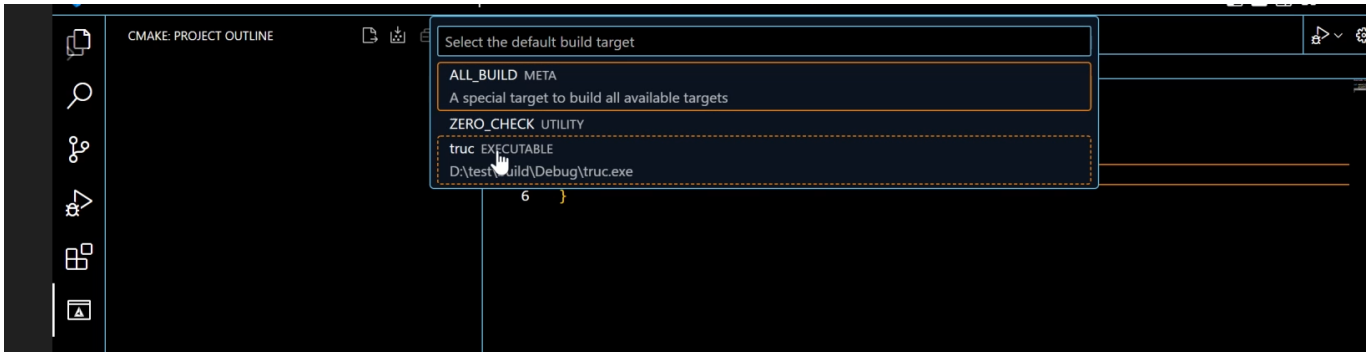

FIGURE 3.1.13 – Step 13 : Cliquer sur truc EXECUTABLE

Step 14 : Cliquer sur Build en basse

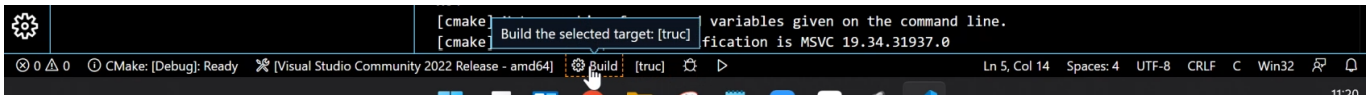

FIGURE 3.1.14 – Step 14 : Cliquer sur Build en basse

Step 15 : Cliquer sur le petit triangle

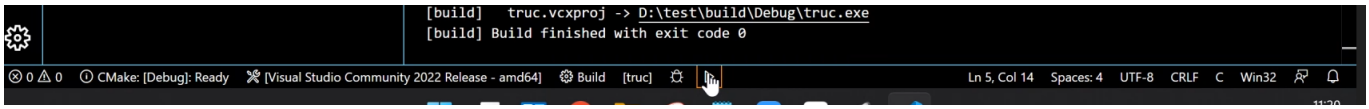

FIGURE 3.1.15 – Step 15 : Cliquer sur le petit triangle

Step 16 : Obtenir le résultat dans TERMINAL

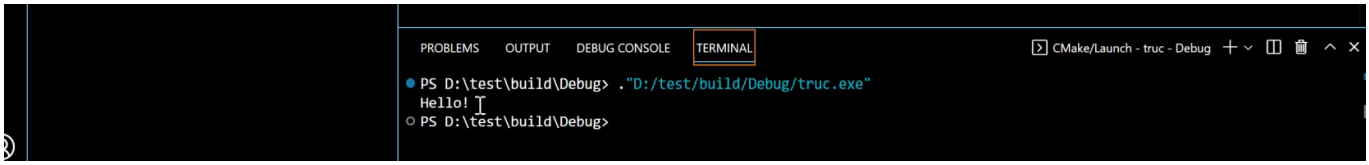

FIGURE 3.1.16 – Step 16 : Obtenir le résultat dans TERMINAL

3.2 input : création l'image à tester et l'image de motif par Photoshop puis enregistrer dans la structure bwimaget en utilisant E3ALoadImage

Listing 2 – input image en utilisant la fonction E3ALoadImage()

```
1 //----------------------image1 : image a tester ----------------------//
2 bwimage_t image1;
3 error_e retval=E3A_OK;
4 retval=E3ALoadImage("../../image/main_tester_input.png", &image1);
```

3.3 conversion image2complexe

Ici on doit convertir des nomber entier de 0 à 255 vers nombre complexe pour faire le FFT à l'étape après, idée est de utiliser la stucture *complex* et *Imagecomplexe*, on ajoute 0 pour la partie imaginaire.

Listing 3 – conversion complexe2image

```
1 Imagecomplexe complex_image1 = conversion_image2complexe(image1);
```

Listing 4 – function conversion_image2complexe

```
1  /*4.function conversion_image2complexe */
2  Imagecomplexe  conversion_image2complexe(bwimage_t image)
3  {
4
5  Imagecomplexe imc;
6
7  imc.width=image.width;
8  imc.height=image.height;
9  imc.crawdata=initialize_tab_c(image.width*image.height);
10
11 int j;
12 for(j=0;j<image.width*image.height;j++){
13     imc.crawdata[j];
14     image.rawdata[j];
15     imc.crawdata[j].real=image.rawdata[j];
16     imc.crawdata[j].imag=0;
17 }
18
19 return imc;
20 }
```

3.4 fft2 : FFT

On fait la FFT en 2 dimension pour passer en fréquencille :

Listing 5 – calculer FFT pour passer image1 en fréquencille

```
1 fft2(complex_image1, 1); // fft
```

On a créé une fonction *fft2()*, dedans on va introduire le code *fourn()* donnée. Aussi on doit ajouter quelques lignes de code (ligne 2 à 5) pour s'adapter notre cas :

Listing 6 – function fft2()

```
1 void fft2(Imagecomplexe imc, int isign){
2         int ndim = 2;
3         unsigned long nn[2];
4         nn[1] = imc.width;
5         nn[0] = imc.height; // P523
6         fourn((float*)imc.crawdata, nn, ndim, isign);
7 }
```

Aussi on doit modifier la fonction *fourn()* donnée. Pour data commence à data[0], on ajoute une ligne de code $data--; nn--;$ (ligne 6) :

Listing 7 – function fourn() un peu de modification

```
1 void fourn(float data[], unsigned long nn[], int ndim, int isign)
2 //attention: data commence a data[0]!!!!!
3 {
4
5         //pour que tableaux commencent a 0
6         data--;  nn--;
```

3.5 Déplacement de motif

Pour l'image 2, soit l'image de motif, on fait la même choose, comme précédante :

Listing 8 – on fait la même choose pour image2

```
1 //--------------------image2 : motif centre-------------------------//
2 bwimage_t image2;
3
```

```
retval=E3ALoadImage("../../image/main_motif_input.png", &image2);

Imagecomplexe complex_image2 = conversion_image2complexe(image2);

fft2(complex_image2, 1); // fft
```

Mais comme notre image2 soit le motif est placé au centre en temporelle, et on voudrait faire la corrélation dans l'étape suivante, il faut le motif soit être périoque en temporelle, donc il faut déplacer le motif centré aux 4 coins. Comme on a en fréquencille, un deplacement soit le retard en temporelle s'agit de faire une multiplication de $e^{-\imath 2\pi\nu a}$ (2.2.3) en fréquencille, on a le code suivant :

Listing 9 – on déplace le motif centré aux 4 coins

```
//------------motif_deplace :  motif deplace au 4 coins ------------//
Imagecomplexe motif_deplace = deplacer_motif(complex_image2);
```

Listing 10 – fonction deplacer_motif()

```
/*11.function deplacer_motif */
Imagecomplexe deplacer_motif(Imagecomplexe motif){

    Imagecomplexe motif_after_deplace;
motif_after_deplace.width=motif.width;
motif_after_deplace.height=motif.height;
motif_after_deplace.crawdata=initialize_tab_c(motif.width*motif.height);

    int j; int omega1; int omega2;

    for(j=0; j<motif_after_deplace.width*motif_after_deplace.height; j++)
        {
        int x = j/motif_after_deplace.height; int y = j%motif_after_deplace.width;
        //int y = j/imc.height; int x = j%imc.width;
        if((0<=x && x< motif_after_deplace.height/2) && (0<=y && y< motif_after_deplace.width
            /2)) {
                omega1=x; omega2=y;
        }
        else if(( motif_after_deplace.height/2 <= x && x < motif_after_deplace.height )  &&
            (0<=y && y< motif_after_deplace.width/2)){
                omega1=x-(signed)motif_after_deplace.height; omega2=y;
        }
        else if((0<=x && x< motif_after_deplace.height/2) && (motif_after_deplace.width/2<= y
            && y< motif_after_deplace.width )){
                omega1=x; omega2=y-motif_after_deplace.width;
        }
        else{
                omega1=x-motif_after_deplace.height; omega2=y-motif_after_deplace.width;
        }

    float OMEGA1=(float)(omega1)/motif_after_deplace.height;
    float OMEGA2=(float)(omega2)/motif_after_deplace.width;
    //float sigma=0.16;
    //float facteur = exp(-(OMEGA1*OMEGA1+OMEGA2*OMEGA2)/(2*sigma*sigma));

    float Re = motif.crawdata[j].real; float Im = motif.crawdata[j].imag;
    float deltax=(float)(motif_after_deplace.height)/2;
    float deltay=(float)(motif_after_deplace.width)/2;
    float angle=2*acos(-1.0)*(OMEGA1*deltax+OMEGA2*deltay);

    (*(motif_after_deplace.crawdata+j)).real=1*(Re*cos(angle)-Im*sin(angle));
    (*(motif_after_deplace.crawdata+j)).imag=1*(Re*sin(angle)+Im*cos(angle));
    }
return motif_after_deplace;
}
```

Remarque 1 : $acos(-1.0) = \pi\ rad$

Remarque 2 : En 1 dimension, la facteur est de $e^{-\imath 2\pi\nu a}$, mais en 2 dimension, si on déplace $f(x+\Delta x, y+\Delta y)$ en fréquencille pour la fonction $F(\nu_x, \nu_y)$, on doit le multilier une facteur comme $e^{-\imath 2\pi(\nu_x\Delta x+\nu_y\Delta y)} = e^{-\imath 2\pi(\Omega_1\Delta x+\Omega_2\Delta y)} = e^{-\imath(\alpha)} = cos(\alpha) + \imath sin(\alpha)$. Ici Ω_1 est OMEGA1 dans notre code ; Ω_2 est OMEGA2 dans notre code ; $\alpha = 2\pi(\Omega_1\Delta x + \Omega_2\Delta y)$

Remarque 3 : Finalement, on utilise ce que l'on parle d'avant :

$$\begin{aligned}\underline{Z_1}\cdot\underline{Z_2} &= \left(\mathfrak{Re}\left(\underline{Z_1}\right)+\imath\mathfrak{Im}\left(\underline{Z_1}\right)\right)\cdot\left(\mathfrak{Re}\left(\underline{Z_2}\right)+\imath\mathfrak{Im}\left(\underline{Z_2}\right)\right)\\ &= \left(\mathfrak{Re}\left(\underline{Z_1}\right)\mathfrak{Re}\left(\underline{Z_2}\right)-\mathfrak{Im}\left(\underline{Z_1}\right)\mathfrak{Im}\left(\underline{Z_2}\right)\right)+\imath\left(\mathfrak{Re}\left(\underline{Z_1}\right)\mathfrak{Im}\left(\underline{Z_2}\right)+\mathfrak{Re}\left(\underline{Z_2}\right)\mathfrak{Im}\left(\underline{Z_1}\right)\right)\end{aligned}$$

pour enregistrer ou dire chercher la partie rélle est la partie imaginaire.

3.6 produit image

Maintenant, on fait le produit image :

Listing 11 – on calcule le produit entre deux image soit la corrélation

```
//------image produit image :  entre image1 et motif_deplace -------//
Imagecomplexe complex_image_produit_image=produit_image_version2(complex_image1, motif_deplace)
    ; // produit_image_version2 : meme principe que test_corr2.c ou il n'y a pas de terme de
    derivee croise
```

Il s'agit un produit en fréquencille (2.4.2), soit

Listing 12 – fonction produit_image_version2()

```
/*7.2.function produit_image_version2 */
Imagecomplexe produit_image_version2(Imagecomplexe complex_image1,  Imagecomplexe
    complex_image2)
{

   Imagecomplexe complex_image_produit_image;

complex_image_produit_image.width=complex_image1.width;
complex_image_produit_image.height=complex_image1.height;
complex_image_produit_image.crawdata=initialize_tab_c(complex_image1.width*complex_image1.
    height);

    int j; int omega1; int omega2;

    for(j=0; j<complex_image_produit_image.width*complex_image_produit_image.height; j++)
        {
        int x = j/complex_image_produit_image.height; int y = j%complex_image_produit_image.
            width;
        //int y = j/imc.height; int x = j%imc.width;
        if((0<=x && x< complex_image_produit_image.height/2) && (0<=y && y<
            complex_image_produit_image.width/2)) {
                omega1=x; omega2=y;
        }
        else if(( complex_image_produit_image.height/2 <= x && x < complex_image_produit_image.
            height )  && (0<=y && y< complex_image_produit_image.width/2)){
                omega1=x-(signed)complex_image_produit_image.height; omega2=y;
        }
        else if((0<=x && x< complex_image_produit_image.height/2) && (
            complex_image_produit_image.width/2<= y && y< complex_image_produit_image.width )){
                omega1=x; omega2=y-complex_image_produit_image.width;
        }
        else{
                omega1=x-complex_image_produit_image.height; omega2=y-
                    complex_image_produit_image.width;
        }

    float OMEGA1=(float)(omega1)/complex_image_produit_image.height;
    float OMEGA2=(float)(omega2)/complex_image_produit_image.width;
    float sigma=0.16;
    float facteur = exp(-(OMEGA1*OMEGA1+OMEGA2*OMEGA2)/(2*sigma*sigma));

    float Re1 = complex_image1.crawdata[j].real; float Im1 = complex_image1.crawdata[j].imag;
    float Re2 = complex_image2.crawdata[j].real; float Im2 = complex_image2.crawdata[j].imag;
    (*(complex_image_produit_image.crawdata+j)).real=facteur*(OMEGA1*OMEGA1+OMEGA2*OMEGA2)*(Re1
        *Re2+Im1*Im2);
    (*(complex_image_produit_image.crawdata+j)).imag=facteur*(OMEGA1*OMEGA1+OMEGA2*OMEGA2)*(-
        Re1*Im2+Re2*Im1);

    }
return complex_image_produit_image;
}
```

Remarque 1 : On toujours utilise

$$\begin{aligned}\underline{Z_1}\cdot\underline{Z_2} &= \left(\mathfrak{Re}\left(\underline{Z_1}\right)+\imath\mathfrak{Im}\left(\underline{Z_1}\right)\right)\cdot\left(\mathfrak{Re}\left(\underline{Z_2}\right)+\imath\mathfrak{Im}\left(\underline{Z_2}\right)\right)\\ &= \left(\mathfrak{Re}\left(\underline{Z_1}\right)\mathfrak{Re}\left(\underline{Z_2}\right)-\mathfrak{Im}\left(\underline{Z_1}\right)\mathfrak{Im}\left(\underline{Z_2}\right)\right)+\imath\left(\mathfrak{Re}\left(\underline{Z_1}\right)\mathfrak{Im}\left(\underline{Z_2}\right)+\mathfrak{Re}\left(\underline{Z_2}\right)\mathfrak{Im}\left(\underline{Z_1}\right)\right)\end{aligned}$$

à enregistrer ou dire chercher la partie rélle est la partie imaginaire.

Remarque 2 : Ici on applique un filtre gausse (2.6.1) pour avoir le mieux produit image en sortie possible ; on doit chisir un sigma de manière à atténuer l'amplitude de la FT en hautes fréquences et lisser un peu les dérivées (pour que ça marche mieux sur des images avec du bruit), dans notre cas ici, on a choisi sigma=0.16.

Remarque 3 : Ici on a aussi fait une opération de gradient pour avoir le contour d'image, on note que ici le gradient est propre, il n'y pas de terme croisé.

Comme ce que on a déjà parler dans (2.2.4), la dérivée en temporlle s'agit une facteur de multiplipication $(\imath 2\pi\nu)^1 = (\imath\omega)$ en fréquencille ; Et si on passe $\mathrm{Corr}(f,g)$ de temporelle à fréquencille, on a $F\cdot G^*$ en fréquencille (2.4.2) ; donc on passe $\mathrm{Corr}(\vec{\nabla}f,\vec{\nabla}g)$ de temporelle à fréquencille, soit $\dfrac{\partial f}{\partial x}\odot\dfrac{\partial g}{\partial x}$ et $\dfrac{\partial f}{\partial y}\odot\dfrac{\partial g}{\partial y}$, en fréquencille on a $(\imath\omega_1 F)\cdot(\imath\omega_1 G)^* = \omega_1^2 FG^*$ et $(\imath\omega_2 F)\cdot(\imath\omega_2 G)^* = \omega_2^2 FG^*$ soit $(\omega_1^2+\omega_2^2)FG^*$ en total.

3.7 fft2 : FFT inverse

Une fois que on fait le produit image en fréquzencille, on fait le FFT inverse pour passer en temporelle, on remarque ici que les data sont toujour en complexe, donc pour avoir une image en sortie, il faut le paseer en entier de 0 à 255, c'est la section suivante où on va parler :

Listing 13 – on calcule le fft inverse

```
//--------------------fft inverse---------------------//
fft2(complex_image_produit_image, -1);
```

Ici dans la fonction *fft2()*, on utilise toujour la même fonction *fourn()* donnée, juste remplace 1 par -1 .

3.8 conversion complexe2image

Pour avoir une image en sortie, il faut le paseer en entier de 0 à 255, on a déjà parler dans la partie théorique (2.5.1).

Listing 14 – on passe de nombre complexe à entier de 0 à 255 pour faire sortir l'image de corrélation

```
//----------------------conversion complexe2image-----------------------//
bwimage_t  int_image = conversion_complexe2image(complex_image_produit_image);
```

Listing 15 – fonction conversion_complexe2image()

```
/*9.function conversion_complexe2image */
bwimage_t  conversion_complexe2image(Imagecomplexe cimage){



    bwimage_t int_image;
    int_image.height = cimage.height;
    int_image.width = cimage.width;
    int_image.data = (unsigned char**)malloc(cimage.height*sizeof(unsigned char**));
    int_image.rawdata=(unsigned char*)malloc((cimage.width) * (cimage.height) * 4 *sizeof(
        unsigned char));

    // Assign pointers
    int y;
    for (int y = 0; y < cimage.height; y++){
        int_image.data[y] = int_image.rawdata+y*int_image.width*sizeof(unsigned char);
    }

int j;
/*
    for(j=0;j<cimage.width*cimage.height;j++){
        *(image+j)=round((*(cimage.crawdata+j)).real);
    }
*/
    int position; /*******/
    float intensity_MAX=(*(cimage.crawdata+0)).real;
    float intensity_MIN=(*(cimage.crawdata+0)).real;
    for(j=0;j<cimage.width*cimage.height;j++){
```

```
28          if((*(cimage.crawdata+j)).real>intensity_MAX){
29              intensity_MAX=(*(cimage.crawdata+j)).real;
30              position = j;/*******/
31          }
32          if((*(cimage.crawdata+j)).real<intensity_MIN){
33              intensity_MIN=(*(cimage.crawdata+j)).real;
34          }
35      }
36      print_tab_c(cimage.crawdata, cimage.width*cimage.height);
37      printf("MAX␣%f␣␣,␣MIN␣%f␣\n", intensity_MAX, intensity_MIN);
38      printf("MAX␣position␣(%d␣,␣%d)␣\n", position/cimage.width, position%cimage.height); /******
            */
39      for(j=0;j<cimage.width*cimage.height;j++){
40          *(int_image.rawdata+j)=round(255*(float)((*(cimage.crawdata+j)).real-intensity_MIN)/(
                intensity_MAX-intensity_MIN));
41          //printf("%u||", *(int_image.rawdata+j));
42      }
43
44  /*
45      for(j=0;j<cimage.width*height;j++){
46          *(int_image.rawdata+j)=*(image+j);
47          //printf("%u||", *(int_image.rawdata+j));
48      }
49  */
50      printf("\n\n");
51      //int_image.rawdata = image;
52
53
54      return int_image;
55
56  }
```

Remarque : Si le tableau data qui n'est pas correctement initialisé, il y a une erreur qui sort. Ce tableau doit contenir les pointeurs vers les lignes de l'image. Dans mon ancien code je alloue le tableau avec int_image.data=(unsigned char**)malloc ... etc mais ça ne suffit pas, les pointeurs dans le tableau ne pointent nulle part. Il faut initialiser le tableau en le reliant au adresses des lignes dans rawdata (voir le code de E3ALoadImage dans pngwrap.c sous le commentaire "assign pointers" : ce sera en gros le même code pour nous).

Donc c'est d'où il vient le code en ligne 13 - 16 nommé Assign pointeur.

3.9 output : E3ADumpImage

On a fait sortir l'image de corrélation, l'enregistier dans le dossier nommé image.

Listing 16 – E3ADumpImage

```
1  //--------------------DumpImage---------------------//
2  E3ADumpImage("../../image/main_tester_output.png", &int_image);
```

4 Visualisation

4.1 Visualisation le gradient : Image gradient

Pour nous, avant d'écrire le gradient propre dans la fonction *conversion complexe2image()*, on a aussi essayer de faire une fonction *gradient_image(Imagecomplexe imc)* pour visualiser qu'est ce que ça se passe pour calculer le gradeint :

Listing 17 – E3ADumpImage

```
1  void gradient_image(Imagecomplexe imc){
2          int j; int omega1; int omega2;
3
4          for(j=0; j<imc.width*imc.height; j++)
5          {
6          int x = j/imc.height; int y = j%imc.width;
7          //int y = j/imc.height; int x = j%imc.width;
8          if((0<=x && x< imc.height/2) && (0<=y && y< imc.width/2)) {
9                  omega1=x; omega2=y;
10         }
11         else if(( imc.height/2 <= x && x < imc.height )  && (0<=y && y< imc.width/2)){
12                 omega1=x-(signed)imc.height; omega2=y;
13         }
14         else if((0<=x && x< imc.height/2) && (imc.width/2<= y && y< imc.width )){
```

```
                omega1=x; omega2=y-imc.width;
        }
        else{
                omega1=x-imc.height; omega2=y-imc.width;
        }
        //      printf("(%d-%f, %d-%f) |", x, OMEGA1, y, OMEGA2);
//      imc.crawdata[j].real=-1*(imc.crawdata[j].imag)*(omega1+omega2);
//      imc.crawdata[j].imag=(imc.crawdata[j].real)*(omega1+omega2);

    /*
    float Re=imc.crawdata[j].real;
        float Im=imc.crawdata[j].imag;
    imc.crawdata[j].real=1*(-1*(Im)*(omega1+omega2));
        imc.crawdata[j].imag=1*((Re)*(omega1+omega2));
        printf("(%d-%d, %d-%d) |", x, omega1, y, omega2);
    */


        float OMEGA1=(float)(omega1)/imc.height;  float OMEGA2=(float)(omega2)/imc.width;
        //float sigma=(abs(OMEGA1)>abs(OMEGA2))?OMEGA1:OMEGA2;
        float sigma=0.16;
        float facteur = exp(-(OMEGA1*OMEGA1+OMEGA2*OMEGA2)/(2*sigma*sigma));
        float Re=imc.crawdata[j].real; float Im=imc.crawdata[j].imag;
    imc.crawdata[j].real=1*(-1*(Im)*facteur*(OMEGA1+OMEGA2));
        imc.crawdata[j].imag=1*((Re)*facteur*(OMEGA1+OMEGA2));
        ///printf("(%d-%f, %d-%f) |", x, OMEGA1, y, OMEGA2);


        }
}
```

Remarque : Ici ce que l'on a fait en temporelle c'est calculer $\mathrm{Corr}((\frac{\partial f}{\partial x}+\frac{\partial f}{\partial y}),(\frac{\partial g}{\partial x}+\frac{\partial g}{\partial y}))$ soit $=\mathrm{Corr}(\frac{\partial f}{\partial x},\frac{\partial g}{\partial x})+\mathrm{Corr}(\frac{\partial f}{\partial y},\frac{\partial g}{\partial y})+\mathrm{Corr}(\frac{\partial f}{\partial x},\frac{\partial g}{\partial y})++\mathrm{Corr}(\frac{\partial f}{\partial y},\frac{\partial g}{\partial x})$ d'où il vient le terme croisé. On va répresenter le grandient avec des termes dans la partie Comparaison des résults, on va voir le contour d'une image obtenu par cette fonction ; et de plus on voit bien qu'il est mieux d'utiliser le gradient propre soit pas de terme croisé.

4.2 Visualisation la corrélation sans faire le gradient

Tout en début, on a fait une teste de sans faire le grandient, pour les images à tester simple (caré blanc par exemple), on voit pas trop qu'est ce qu'il se passe ; mais si l'on utilise des images plus complexe, le résutat qui sort n'est pas bien, donc il est nécessaire de calculer le grandient d'image pour faire la corrélation entre deux contour, on va aussi le montrer dans la partie Comparaison des résults.

4.3 Visualisation le sepctre de FFT en utilisant MATLAB

On a aussi fait un code en MATLAB pour visualiser le sepctre de FFT, voici le spectre est le code utilisé en MATLAB :

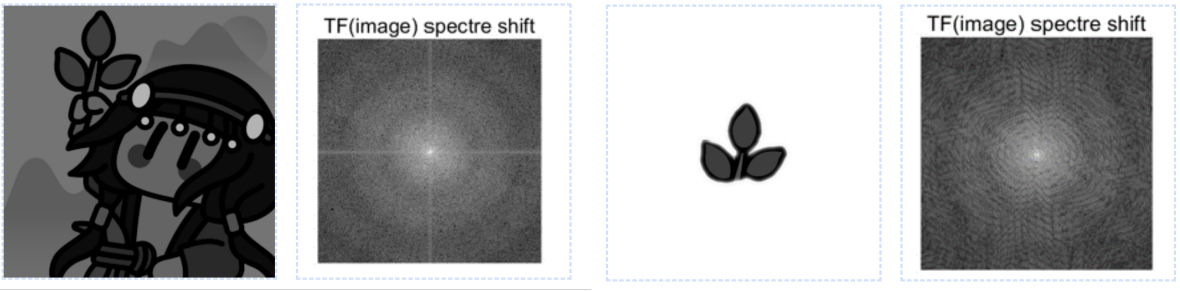

FIGURE 4.3.1 – Visualisation le sepctre de FFT en utilisant MATLAB

Listing 18 – visualiser le sepctre de FFT

```
clear
clc
img=imread('m.png');
subplot(2,2,1);imshow(img);title('image');
f=rgb2gray(img);
F=fft2(f);
F1=log(abs(F)+1);
```

```
subplot(2,2,2);imshow(F1,[]);title('TF(image) spectre');
Fs=fftshift(F);
S=log(abs(Fs)+1);
subplot(2,2,3);imshow(S,[]);title('TF(image) spectre shift');
fr=real(ifft2(ifftshift(Fs)));
ret=im2uint8(mat2gray(fr));
subplot(2,2,4);imshow(ret),title('TF-1[TF(image)]');
```

5 Comparaison des résults

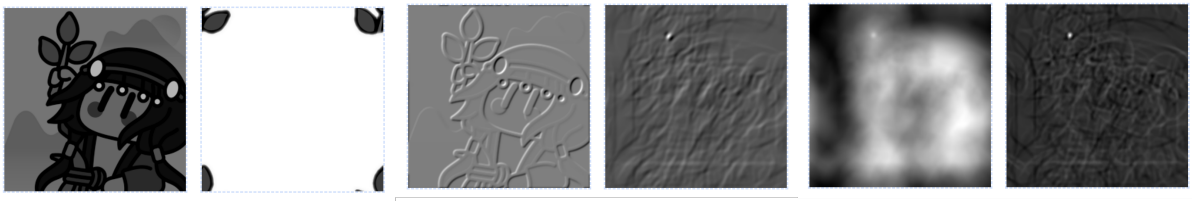

FIGURE 5.0.1 – Comparaison des résults

- image 1 est notre image à tester
- image 2 est le motif
- image 3 est la visualisation de la fonction *gradient image(Imagecomplexe imc)*, il calcule le grandient avec des terme croisé dedans
- image 4 est le corrélation entre image1 et image2 en utilisant la fonction *gradient image(Imagecomplexe imc)*
- image 5 est la visualisation de la corrélation où on n'ai pas fait le grandient image, donc on ne peut pas reconnait des formes par corrélation d'images
- image 6 est le résultat de notre code final, il n'y a pas de terme croisé dans dans la fonction *produit image version2()*. En comparant avec impage 4, image 6 est plus idéal, on voir bien la différence entre le noir profond et blanc clair.

6 Conclusion

Pendant notre projet :

On apprend à écrire le code dans Visual Studio Code ; on joue avec les structures à enregistrer les data d'image ;

On a révisé beaucoup de outils mathélatique telles que Racine n-ième de l'unité, les propriétés dev Transformation de Fourier, Théorème de la Correlation ...

On a bien réussi notre code : on envoie une image à tester et une impage de motif (Les trèfles) dans le programme main.c, on a bien vu il y a des picel blanc, soit intensité intense dans l'image en sortie, le code donc peut bien reconnait la forme de motif par corrélation d'images et prédire où il est.

Lots of thanks for Pawel Wzietek ; Dan Pineau ; Christophe Vignat

01/04/2023

à Orsay

7 ReadME

Dans le dossier .zip, il y a plusieurs de fichier .c, voici je présente à quoi se sert :

test1_projet.c C'est notre prémière version de code : on utilise des tableux à enregistrer les data, ce n'est pas intélligent, donc on a modifié après en utilisant la stucture *Imagecomplexe* à enregistrer les data ; De plus on n'a pas fait le gradient image, donc si on teste avec les images compliquées, on n'arrive pas à obtenir le résultat ideal.

test1_corr_gradient1.c Comme si on traite les images compliquées, on n'arrive pas à obtenir le résultat idéal, maintenant on crée la fonction *gradient_image()* pour laisser le contour (Attension : il y a des termes croisés, on va ameliorer dans test3_corr2). Toujour noté que ici on utilise des tableux à enregistrer les data, on va ameliorer depuis test2 .

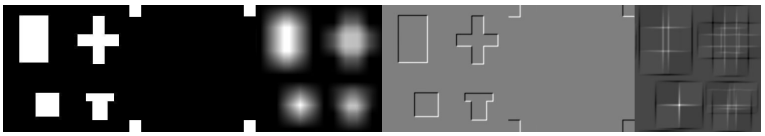

FIGURE 7.0.1 – Image référence simple : de 1 à 3 viennent de test1_projet.c où on n'est pas fait le gradient ; de 4 à 6 viennent de test1_corr_gradient1.c, où on calcule le gradient avec terme croisé

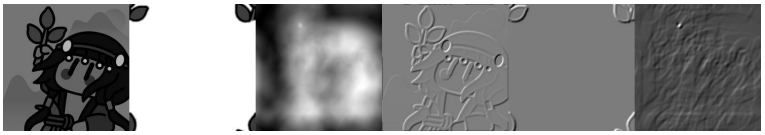

FIGURE 7.0.2 – Image reference complexe : de 1 à 3 viennent de test1_projet.c où on n'est pas fait le gradient ; de 4 à 6 viennent de test1_corr_gradient1.c, où on calcule le gradient avec terme croisé

test2_grandient_image.c On utilise la structure *Imagecomplexe ;* à enregistrer les données depuis maintenat. Ce code calcule le gradient d'image avec terme croisé, juste pour visualiser le gradient image, dans test3_corr2, on va présenter une autre manière où il n'y a pas de terme croisé :

FIGURE 7.0.3 – Gradient image : 1 est l'image référence ; 2 on calcule le gradient avec terme croisé

test3_corr1.c Une fois que on a la fonction *gradient_image()*, on essaie le prodoit d'image en utilisant le gradient avec terme croisé dedans bien sur.

FIGURE 7.0.4 – Corrélation méthode 1 : 1 est l'image référence ; 2 est le motif ; 3 est le output

test3_corr2.c Dans la fonction *produit_image_version2()*, on fait l'opération gradient directement, le gradient ici est propre, il n'y a pas de terme croisé. Le résultat est mieux (le noir est plus noire, le blanc est plus blanc)

FIGURE 7.0.5 – Corrélation méthode 2 : 1 est l'image référence ; 2 est le motif ; 3 est le output

test4_deplace_image.c Pour évider chaque fois on déplace le motif à 4 coins en mains, on crée la fonction *deplacer_motif()* :

FIGURE 7.0.6 – Déplacer motif : 1 est motif centré ; 2 est motif déplacé

main.c Notre code version finale : intput l'image référence et le motif centré, voici le résultat

FIGURE 7.0.7 – Déplacer motif : 1 est image référence ; 2 est motif centré ; 3 est output

8 Annexe

Listing 19 – CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20)
project(test_png)
set(CMAKE_C_STANDARD 99)

set(LIBDEPS "${CMAKE_SOURCE_DIR}/deps")

#set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/bin)
#SET( CMAKE_RUNTIME_OUTPUT_DIRECTORY_DEBUG "${CMAKE_RUNTIME_OUTPUT_DIRECTORY}")
#SET( CMAKE_RUNTIME_OUTPUT_DIRECTORY_RELEASE "${CMAKE_RUNTIME_OUTPUT_DIRECTORY}")

file(COPY "${CMAKE_SOURCE_DIR}/lena.png" DESTINATION "${CMAKE_BINARY_DIR}")

# Find the libpng library
#find_package(PNG REQUIRED)

include_directories( "${LIBDEPS}/include")

# Add an executable target
#add_executable(test_png pngtest.c pngwrap.c)
#add_executable(test_1 structure_complexe.c maintest1.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c projet.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c test1_projet.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c test1_grandient_image.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c test1_corr_gradient1.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c test2_grandient_image.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c test3_corr1.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c test3_corr2.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c test4_deplace_image.c)
add_executable(test_png imagec.c pngwrap.c fft2.c main.c)
##add_executable(test_png imagec.c pngwrap.c fft2.c test0.c)

# Link the libpng library to the test_png target
#target_link_libraries(test_png PNG::PNG)
target_link_libraries(test_png "${LIBDEPS}/lib/libpng16_static.lib" "${LIBDEPS}/lib/zlibstatic.
    lib")
```

Listing 20 – projet.h

```
#include "pngwrap.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


/*1.struct for complex */
```

```
typedef struct
{
    float real;
    float imag;
} complex;

/*2.struct for Imagecomplexe*/
typedef struct
{
    unsigned int width;
    unsigned int height;
     //complex **cdata; /* Access rgba pixels as image.data[row][col*4+i] where i=0, 1, 2, 3 for R, G, B and alpha */
     complex *crawdata;
} Imagecomplexe;

/*3.function for initialize a tab_complex and release the tab_c and print the tab_c */
complex *initialize_tab_c(int column1d);

void release_tab_c(complex *arr1d);

void print_tab_c(complex *arr1d, int column1d);

/*4.function conversion_image2complexe */
Imagecomplexe  conversion_image2complexe(bwimage_t image);

/*5.function fourn */
void fourn(float data[], unsigned long nn[], int ndim, int isign);

/*6.function fourier */
void fft2(Imagecomplexe imc, int isign);


/*7.function produit_image */
//void produit_image(Imagecomplexe complex_image1,  Imagecomplexe complex_image2, complex * complex_image_produit_image);
Imagecomplexe produit_image(Imagecomplexe complex_image1,  Imagecomplexe complex_image2);


/*8.function for initialize a tab_int and release the tab_i and print the tab_i */
int *initialize_tab_i(int ncolumns);

void release_tab_i(int *m);

void print_tab_i(int *m, int c);


/*9.function conversion_complexe2image */
bwimage_t  conversion_complexe2image(Imagecomplexe cimage);

/*10.function conversion_complexe2image */
//bwimage_t conversion_complexe2image(int width, int height, complex * complex_image_correlation_simple, int *image);

/*7.2.function produit_image_version2 */
Imagecomplexe produit_image_version2(Imagecomplexe complex_image1,  Imagecomplexe complex_image2);

/*11.function deplacer_motif */
Imagecomplexe deplacer_motif(Imagecomplexe motif);
```

Listing 21 – fft2.c

```
#include <stdio.h> //
#include <stdlib.h> //
#include "projet.h" //

#include <math.h>
#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr

void fourn(float data[], unsigned long nn[], int ndim, int isign)
//attention: data commence a data[0]!!!!!
{

```

```
12          //pour que tableaux commencent a 0
13          data--;  nn--;
14
15          int idim;
16          unsigned long i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
17          unsigned long ibit,k1,k2,n,nprev,nrem,ntot;
18          float tempi,tempr;
19          double theta,wi,wpi,wpr,wr,wtemp;
20
21          for (ntot=1,idim=1;idim<=ndim;idim++)
22                  ntot *= nn[idim];
23          nprev=1;
24          for (idim=ndim;idim>=1;idim--) {
25                  n=nn[idim];
26                  nrem=ntot/(n*nprev);
27                  ip1=nprev << 1;
28                  ip2=ip1*n;
29                  ip3=ip2*nrem;
30                  i2rev=1;
31                  for (i2=1;i2<=ip2;i2+=ip1) {
32                          if (i2 < i2rev) {
33                                  for (i1=i2;i1<=i2+ip1-2;i1+=2) {
34                                          for (i3=i1;i3<=ip3;i3+=ip2) {
35                                                  i3rev=i2rev+i3-i2;
36                                                  SWAP(data[i3],data[i3rev]);
37                                                  SWAP(data[i3+1],data[i3rev+1]);
38                                          }
39                                  }
40                          }
41                          ibit=ip2 >> 1;
42                          while (ibit >= ip1 && i2rev > ibit) {
43                                  i2rev -= ibit;
44                                  ibit >>= 1;
45                          }
46                          i2rev += ibit;
47                  }
48                  ifp1=ip1;
49                  while (ifp1 < ip2) {
50                          ifp2=ifp1 << 1;
51                          theta=isign*6.28318530717959/(ifp2/ip1);
52                          wtemp=sin(0.5*theta);
53                          wpr = -2.0*wtemp*wtemp;
54                          wpi=sin(theta);
55                          wr=1.0;
56                          wi=0.0;
57                          for (i3=1;i3<=ifp1;i3+=ip1) {
58                                  for (i1=i3;i1<=i3+ip1-2;i1+=2) {
59                                          for (i2=i1;i2<=ip3;i2+=ifp2) {
60                                                  k1=i2;
61                                                  k2=k1+ifp1;
62                                                  tempr=(float)wr*data[k2]-(float)wi*data[k2+1];
63                                                  tempi=(float)wr*data[k2+1]+(float)wi*data[k2];
64                                                  data[k2]=data[k1]-tempr;
65                                                  data[k2+1]=data[k1+1]-tempi;
66                                                  data[k1] += tempr;
67                                                  data[k1+1] += tempi;
68                                          }
69                                  }
70                                  wr=(wtemp=wr)*wpr-wi*wpi+wr;
71                                  wi=wi*wpr+wtemp*wpi+wi;
72                          }
73                          ifp1=ifp2;
74                  }
75                  nprev *= n;
76          }
77  }
78  // #undef SWAP
79
80  void fft2(Imagecomplexe imc, int isign){
81          int ndim = 2;
82          unsigned long nn[2];
83          nn[1] = imc.width;
84          nn[0] = imc.height; // P523
85          fourn((float*)imc.crawdata, nn, ndim, isign);
```

```
}

void gradient_image(Imagecomplexe imc){
        int j; int omega1; int omega2;

        for(j=0; j<imc.width*imc.height; j++)
        {
        int x = j/imc.height; int y = j%imc.width;
        //int y = j/imc.height; int x = j%imc.width;
        if((0<=x && x< imc.height/2) && (0<=y && y< imc.width/2)) {
                omega1=x; omega2=y;
        }
        else if(( imc.height/2 <= x && x < imc.height )  && (0<=y && y< imc.width/2)){
                omega1=x-(signed)imc.height; omega2=y;
        }
        else if((0<=x && x< imc.height/2) && (imc.width/2<= y && y< imc.width )){
                omega1=x; omega2=y-imc.width;
        }
        else{
                omega1=x-imc.height; omega2=y-imc.width;
        }
        //      printf("(%d-%f, %d-%f) |", x, OMEGA1, y, OMEGA2);
//      imc.crawdata[j].real=-1*(imc.crawdata[j].imag)*(omega1+omega2);
//      imc.crawdata[j].imag=(imc.crawdata[j].real)*(omega1+omega2);

    /*
    float Re=imc.crawdata[j].real;
        float Im=imc.crawdata[j].imag;
    imc.crawdata[j].real=1*(-1*(Im)*(omega1+omega2));
        imc.crawdata[j].imag=1*((Re)*(omega1+omega2));
        printf("(%d-%d, %d-%d) |", x, omega1, y, omega2);
    */


        float OMEGA1=(float)(omega1)/imc.height;  float OMEGA2=(float)(omega2)/imc.width;
        //float sigma=(abs(OMEGA1)>abs(OMEGA2))?OMEGA1:OMEGA2;
        float sigma=0.16;
        float facteur = exp(-(OMEGA1*OMEGA1+OMEGA2*OMEGA2)/(2*sigma*sigma));
        float Re=imc.crawdata[j].real; float Im=imc.crawdata[j].imag;
    imc.crawdata[j].real=1*(-1*(Im)*facteur*(OMEGA1+OMEGA2));
        imc.crawdata[j].imag=1*((Re)*facteur*(OMEGA1+OMEGA2));
        ///printf("(%d-%f, %d-%f) |", x, OMEGA1, y, OMEGA2);


        }
}
```

Listing 22 – imagec.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "projet.h"

/*3.function for initialize a tab_complex and release the tab_c and print the tab_c */
/*function print initialize a tab_complex*/
complex *initialize_tab_c(int column1d){
        int j;
        complex *arr1d = (complex*)malloc(column1d * sizeof(complex));

        for(j=0; j<column1d ;j++){
        (*(arr1d+j)).real=j+0.1; //initialiser a initial_value
        (*(arr1d+j)).imag=j+0.2;
    }
    return arr1d;
}

/*fonction release the tab_c*/
void release_tab_c(complex *arr1d){
    free(arr1d);
}

/*fonction print the tab_c*/
void print_tab_c(complex *arr1d, int column1d){
```

```
        int j;
        for (j=0; j<column1d; j++){
                printf("|(%f,␣%f)␣",(*(arr1d+j)).real, (*(arr1d+j)).imag);
        }
    printf("\n\n");
}

/*4.function conversion_image2complexe */
Imagecomplexe  conversion_image2complexe(bwimage_t image)
{

Imagecomplexe imc;

imc.width=image.width;
imc.height=image.height;
imc.crawdata=initialize_tab_c(image.width*image.height);

int j;
for(j=0;j<image.width*image.height;j++){
    imc.crawdata[j];
    image.rawdata[j];
    imc.crawdata[j].real=image.rawdata[j];
    imc.crawdata[j].imag=0;
}

return imc;
}

/*7.function produit_image */
Imagecomplexe produit_image(Imagecomplexe complex_image1,  Imagecomplexe complex_image2)
{

   Imagecomplexe complex_image_produit_image;



complex_image_produit_image.width=complex_image1.width;
complex_image_produit_image.height=complex_image1.height;
complex_image_produit_image.crawdata=initialize_tab_c(complex_image1.width*complex_image1.
    height);

    int j;
    for(j=0;j<complex_image1.width*complex_image2.height;j++){
        float Re1 = complex_image1.crawdata[j].real; float Im1 = complex_image1.crawdata[j].
            imag;
        float Re2 = complex_image2.crawdata[j].real; float Im2 = complex_image2.crawdata[j].
            imag;
        (*(complex_image_produit_image.crawdata+j)).real=Re1*Re2+Im1*Im2;
        (*(complex_image_produit_image.crawdata+j)).imag=-Re1*Im2+Re2*Im1;
    }
return complex_image_produit_image;
}


/*8.function for initialize a tab_int and release the tab_i and print the tab_i */
/*function print initialize a tab_int*/
int *initialize_tab_i(int ncolumns){
        int j;
        int *m = (int*)malloc(ncolumns * sizeof(int));

        for(j=0; j<ncolumns ;j++){
        *(m+j)=j+1; //initialiser a initial_value
        //printf("m[%d]=%d\n",j,*(m+j));
    }
    return m;
}

/*fonction release the tab_i*/
void release_tab_i(int *m){
    free(m);
}

/*fonction print tab_i*/
void print_tab_i(int *m, int c){
```

```
97          int j;
98          for (j=0; j<c; j++){
99                  printf("|%d",*(m+j));
100         }
101     printf("\n\n");
102 }
103
104 /*9.function conversion_complexe2image */
105 bwimage_t  conversion_complexe2image(Imagecomplexe cimage){
106
107
108
109     bwimage_t int_image;
110     int_image.height = cimage.height;
111     int_image.width = cimage.width;
112     int_image.data = (unsigned char**)malloc(cimage.height*sizeof(unsigned char**));
113     int_image.rawdata=(unsigned char*)malloc((cimage.width) * (cimage.height) * 4 *sizeof(
        unsigned char));
114
115     // Assign pointers
116     int y;
117     for (int y = 0; y < cimage.height; y++){
118         int_image.data[y] = int_image.rawdata+y*int_image.width*sizeof(unsigned char);
119     }
120
121 int j;
122 /*
123     for(j=0;j<cimage.width*cimage.height;j++){
124         *(image+j)=round((*(cimage.crawdata+j)).real);
125     }
126 */
127     int position; /*******/
128     float intensity_MAX=(*(cimage.crawdata+0)).real;
129     float intensity_MIN=(*(cimage.crawdata+0)).real;
130     for(j=0;j<cimage.width*cimage.height;j++){
131         if((*(cimage.crawdata+j)).real>intensity_MAX){
132             intensity_MAX=(*(cimage.crawdata+j)).real;
133             position = j;/*******/
134         }
135         if((*(cimage.crawdata+j)).real<intensity_MIN){
136             intensity_MIN=(*(cimage.crawdata+j)).real;
137         }
138     }
139     print_tab_c(cimage.crawdata, cimage.width*cimage.height);
140     printf("MAX␣%f␣␣,␣MIN␣%f␣\n", intensity_MAX, intensity_MIN);
141     printf("MAX␣position␣(%d␣,␣%d)␣\n", position/cimage.width, position%cimage.height); /******
        */
142     for(j=0;j<cimage.width*cimage.height;j++){
143         *(int_image.rawdata+j)=round(255*(float)((*(cimage.crawdata+j)).real-intensity_MIN)/(
            intensity_MAX-intensity_MIN));
144         //printf("%u||", *(int_image.rawdata+j));
145     }
146
147 /*
148     for(j=0;j<cimage.width*height;j++){
149         *(int_image.rawdata+j)=*(image+j);
150         //printf("%u||", *(int_image.rawdata+j));
151     }
152 */
153     printf("\n\n");
154     //int_image.rawdata = image;
155
156
157     return int_image;
158
159 }
160
161 /*10.function conversion_complexe2image */
162 /*
163 bwimage_t conversion_complexe2image(int width, int height, complex *complex_image_produit_image
        , int *image){
164     Imagecomplexe cimage;
165     cimage.height = height;
166     cimage.width = width;
```

```
167     cimage.crawdata = complex_image_produit_image;
168     //fft2(cimage, -1);
169 
170     bwimage_t int_image;
171     int_image.height = height;
172     int_image.width = width;
173     int_image.data = (unsigned char**)malloc(height*sizeof(unsigned char**));
174     int_image.rawdata=(unsigned char*)malloc((width) * (height) * 4 *sizeof(unsigned char));
175 
176     // Assign pointers
177     int y;
178     for (int y = 0; y < height; y++){
179         int_image.data[y] = int_image.rawdata+y*int_image.width*sizeof(unsigned char);
180     }
181 
182 int j;
183 //
184     for(j=0;j<width*height;j++){
185         *(image+j)=round((*(complex_image_produit_image+j)).real);
186     }
187 //
188     float intensity_MAX=(*(complex_image_produit_image+0)).real;
189     float intensity_MIN=(*(complex_image_produit_image+0)).real;
190     for(j=0;j<width*height;j++){
191         if((*(complex_image_produit_image+j)).real>intensity_MAX){
192             intensity_MAX=(*(complex_image_produit_image+j)).real;
193         }
194         if((*(complex_image_produit_image+j)).real<intensity_MIN){
195             intensity_MIN=(*(complex_image_produit_image+j)).real;
196         }
197     }
198     print_tab_c(complex_image_produit_image, width*height);
199     printf("MAX %f   , MIN %f \n", intensity_MAX, intensity_MIN);
200 
201     for(j=0;j<width*height;j++){
202         *(image+j)=round(255*(float)((*(complex_image_produit_image+j)).real-intensity_MIN)/(
            intensity_MAX-intensity_MIN));
203     }
204 
205     for(j=0;j<width*height;j++){
206         *(int_image.rawdata+j)=*(image+j);
207         //printf("%u||", *(int_image.rawdata+j));
208     }
209     printf("\n\n");
210     //int_image.rawdata = image;
211 
212 *
213 
214     return int_image;
215 
216 }
217 */
218 
219 /*7.2.function produit_image_version2 */
220 Imagecomplexe produit_image_version2(Imagecomplexe complex_image1,  Imagecomplexe
    complex_image2)
221 {
222 
223    Imagecomplexe complex_image_produit_image;
224 
225 complex_image_produit_image.width=complex_image1.width;
226 complex_image_produit_image.height=complex_image1.height;
227 complex_image_produit_image.crawdata=initialize_tab_c(complex_image1.width*complex_image1.
    height);
228 
229     int j; int omega1; int omega2;
230 
231     for(j=0; j<complex_image_produit_image.width*complex_image_produit_image.height; j++)
232         {
233         int x = j/complex_image_produit_image.height; int y = j%complex_image_produit_image.
            width;
234         //int y = j/imc.height; int x = j%imc.width;
235         if((0<=x && x< complex_image_produit_image.height/2) && (0<=y && y<
            complex_image_produit_image.width/2)) {
```

```
                omega1=x; omega2=y;
        }
        else if(( complex_image_produit_image.height/2 <= x && x < complex_image_produit_image.height )  && (0<=y && y< complex_image_produit_image.width/2)){
                omega1=x-(signed)complex_image_produit_image.height; omega2=y;
        }
        else if((0<=x && x< complex_image_produit_image.height/2) && ( complex_image_produit_image.width/2<= y && y< complex_image_produit_image.width )){
                omega1=x; omega2=y-complex_image_produit_image.width;
        }
        else{
                omega1=x-complex_image_produit_image.height; omega2=y-complex_image_produit_image.width;
        }

    float OMEGA1=(float)(omega1)/complex_image_produit_image.height;
    float OMEGA2=(float)(omega2)/complex_image_produit_image.width;
    float sigma=0.16;
    float facteur = exp(-(OMEGA1*OMEGA1+OMEGA2*OMEGA2)/(2*sigma*sigma));

    float Re1 = complex_image1.crawdata[j].real; float Im1 = complex_image1.crawdata[j].imag;
    float Re2 = complex_image2.crawdata[j].real; float Im2 = complex_image2.crawdata[j].imag;
    (*(complex_image_produit_image.crawdata+j)).real=facteur*(OMEGA1*OMEGA1+OMEGA2*OMEGA2)*(Re1*Re2+Im1*Im2);
    (*(complex_image_produit_image.crawdata+j)).imag=facteur*(OMEGA1*OMEGA1+OMEGA2*OMEGA2)*(-Re1*Im2+Re2*Im1);

    }
return complex_image_produit_image;
}


/*11.function deplacer_motif */
Imagecomplexe deplacer_motif(Imagecomplexe motif){

    Imagecomplexe motif_after_deplace;
motif_after_deplace.width=motif.width;
motif_after_deplace.height=motif.height;
motif_after_deplace.crawdata=initialize_tab_c(motif.width*motif.height);

    int j; int omega1; int omega2;

    for(j=0; j<motif_after_deplace.width*motif_after_deplace.height; j++)
        {
        int x = j/motif_after_deplace.height; int y = j%motif_after_deplace.width;
        //int y = j/imc.height; int x = j%imc.width;
        if((0<=x && x< motif_after_deplace.height/2) && (0<=y && y< motif_after_deplace.width/2)) {
                omega1=x; omega2=y;
        }
        else if(( motif_after_deplace.height/2 <= x && x < motif_after_deplace.height )  && (0<=y && y< motif_after_deplace.width/2)){
                omega1=x-(signed)motif_after_deplace.height; omega2=y;
        }
        else if((0<=x && x< motif_after_deplace.height/2) && (motif_after_deplace.width/2<= y && y< motif_after_deplace.width )){
                omega1=x; omega2=y-motif_after_deplace.width;
        }
        else{
                omega1=x-motif_after_deplace.height; omega2=y-motif_after_deplace.width;
        }

    float OMEGA1=(float)(omega1)/motif_after_deplace.height;
    float OMEGA2=(float)(omega2)/motif_after_deplace.width;
    //float sigma=0.16;
    //float facteur = exp(-(OMEGA1*OMEGA1+OMEGA2*OMEGA2)/(2*sigma*sigma));

    float Re = motif.crawdata[j].real; float Im = motif.crawdata[j].imag;
    float deltax=(float)(motif_after_deplace.height)/2;
    float deltay=(float)(motif_after_deplace.width)/2;
    float angle=2*acos(-1.0)*(OMEGA1*deltax+OMEGA2*deltay);

    (*(motif_after_deplace.crawdata+j)).real=1*(Re*cos(angle)-Im*sin(angle));
    (*(motif_after_deplace.crawdata+j)).imag=1*(Re*sin(angle)+Im*cos(angle));
```

```
    }
return motif_after_deplace;
}
```

Listing 23 – test2_grandient_image.c

```
#include <stdio.h>
#include <stdlib.h>
#include "projet.h"
#define case 55

int main(){
//---------------------image1----------------------//
bwimage_t image1;
error_e retval=E3A_OK;
retval=E3ALoadImage("../../image/test2_gradient_image1_portait2_input.png", &image1);
Imagecomplexe complex_image1 = conversion_image2complexe(image1);
printf("(image1.width, image1.height) = (%d, %d)\n", image1.width, image1.height);
printf("(complex_image1.width, complex_image1.height) == (%d, %d)\n", complex_image1.width,
    complex_image1.height);
printf("%d - ieme element complexe de vector=(%f, %f)\n", case, complex_image1.crawdata[case
    -1].real, complex_image1.crawdata[case-1].imag);
fft2(complex_image1, 1); // fft
gradient_image(complex_image1);
printf("%d - ieme element complexe de vector apres fft=(%f, %f)\n", case, complex_image1.
    crawdata[case-1].real, complex_image1.crawdata[case-1].imag);
//printf("3-ieme element complexe de vector apres fft -1 =(%f , %f)\n", complex_image1.crawdata
    [2].real, complex_image1.crawdata[2].imag);


//---------------------fft inverse----------------------//
fft2(complex_image1, -1);

//---------------------conversion complexe2image----------------------//
bwimage_t  int_image = conversion_complexe2image(complex_image1);


E3ADumpImage("../../image/test2_gradient_image2_portait2_output.png", &int_image);

//release_tab_c(complex_image_produit_image);
//release_tab_i(image);
return 0;
}
```

Listing 24 – test3_corr1.c

```
#include <stdio.h>
#include <stdlib.h>
#include "projet.h"
#define case 55

int main(){
//---------------------image1----------------------//
bwimage_t image1;
error_e retval=E3A_OK;

retval=E3ALoadImage("../../image/test3_corr1_image1_tester_input.png", &image1);

Imagecomplexe complex_image1 = conversion_image2complexe(image1);

printf("(image1.width, image1.height) = (%d, %d)\n", image1.width, image1.height);
printf("(complex_image1.width, complex_image1.height) == (%d, %d)\n", complex_image1.width,
    complex_image1.height);
printf("%d - ieme element complexe de vector=(%f, %f)\n", case, complex_image1.crawdata[case
    -1].real, complex_image1.crawdata[case-1].imag);
fft2(complex_image1, 1); // fft
gradient_image(complex_image1); /// grad

printf("%d - ieme element complexe de vector apres fft=(%f, %f)\n", case, complex_image1.
    crawdata[case-1].real, complex_image1.crawdata[case-1].imag);
//fft2(complex_image1, -1); // fft -1
//printf("3-ieme element complexe de vector apres fft -1 =(%f , %f)\n", complex_image1.crawdata
    [2].real, complex_image1.crawdata[2].imag);
```

```
//--------------------image2---------------------//
bwimage_t image2;

retval=E3ALoadImage("../../image/test3_corr1_image2_motif_intput.png", &image2);

Imagecomplexe complex_image2 = conversion_image2complexe(image2);

printf("(image2.width, image2.height)= (%d , %d)\n", image2.width, image2.height);
printf("(complex_image2.width, complex_image2.height) == (%d , %d)\n", complex_image2.width,
    complex_image2.height);
printf("%d -ieme element complexe de vector=(%f , %f)\n", case, complex_image2.crawdata[case
    -1].real, complex_image2.crawdata[case-1].imag);
fft2(complex_image2, 1); // fft
printf("%d -ieme element complexe de vector apres fft=(%f , %f)\n", case, complex_image2.
    crawdata[case-1].real, complex_image2.crawdata[case-1].imag);
gradient_image(complex_image2); /// grad
//--------------------image produit image---------------------//
Imagecomplexe complex_image_produit_image=produit_image(complex_image1, complex_image2);
printf("%d -ieme element complexe de vector apres produit=(%f , %f)\n", case, (*(
    complex_image_produit_image.crawdata+case-1)).real, (*(complex_image_produit_image.crawdata
    +case-1)).imag);

//--------------------fft inverse---------------------//
fft2(complex_image_produit_image, -1);

//--------------------conversion complexe2image---------------------//
bwimage_t  int_image = conversion_complexe2image(complex_image_produit_image);

/*
int *image = initialize_tab_i(complex_image1.width*complex_image1.height);
print_tab_i(image, 64);
print_tab_i(image, 64);

bwimage_t int_image = conversion_complexe2image(complex_image1.width, complex_image1.height,
    complex_image_produit_image.crawdata, image);
*/

//--------------------DumpImage---------------------//
E3ADumpImage("../../image/test3_corr1_image3_tester_output.png", &int_image);

release_tab_c(complex_image_produit_image.crawdata);
//release_tab_i(image); ?
return 0;

/*
int *image = initialize_tab_i(complex_image1.width*complex_image1.height);
print_tab_i(image, 64);
bwimage_t int_image = conversion_complexe2image(complex_image1.width, complex_image1.height,
    complex_image_produit_image, image);
print_tab_i(image, 64);


E3ADumpImage("../../image/output.png", &int_image);

release_tab_c(complex_image_produit_image);
release_tab_i(image);
return 0;
*/
}
```

Listing 25 – test3_corr2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "projet.h"
#define case 55

int main(){
//--------------------image1---------------------//
bwimage_t image1;
error_e retval=E3A_OK;

retval=E3ALoadImage("../../image/test3_corr2_image1_tester_input.png", &image1);

```

```
Imagecomplexe complex_image1 = conversion_image2complexe(image1);

printf("(image1.width, image1.height) = (%d, %d)\n", image1.width, image1.height);
printf("(complex_image1.width, complex_image1.height) == (%d, %d)\n", complex_image1.width, complex_image1.height);
printf("%d-ieme element complexe de vector=(%f, %f)\n", case, complex_image1.crawdata[case-1].real, complex_image1.crawdata[case-1].imag);
fft2(complex_image1, 1); // fft
//gradient_image(complex_image1); /// grad

printf("%d-ieme element complexe de vector apres fft=(%f, %f)\n", case, complex_image1.crawdata[case-1].real, complex_image1.crawdata[case-1].imag);
//fft2(complex_image1, -1); // fft -1
//printf("3-ieme element complexe de vector apres fft -1 =(%f , %f)\n", complex_image1.crawdata[2].real, complex_image1.crawdata[2].imag);
//--------------------image2---------------------//
bwimage_t image2;

retval=E3ALoadImage("../../image/test3_corr2_image2_motif_input.png", &image2);

Imagecomplexe complex_image2 = conversion_image2complexe(image2);

printf("(image2.width, image2.height)=(%d, %d)\n", image2.width, image2.height);
printf("(complex_image2.width, complex_image2.height) == (%d, %d)\n", complex_image2.width, complex_image2.height);
printf("%d-ieme element complexe de vector=(%f, %f)\n", case, complex_image2.crawdata[case-1].real, complex_image2.crawdata[case-1].imag);
fft2(complex_image2, 1); // fft
printf("%d-ieme element complexe de vector apres fft=(%f, %f)\n", case, complex_image2.crawdata[case-1].real, complex_image2.crawdata[case-1].imag);
//gradient_image(complex_image2); /// grad
//--------------------image produit image---------------------//
Imagecomplexe complex_image_produit_image=produit_image_version2(complex_image1, complex_image2);
printf("%d-ieme element complexe de vector apres produit=(%f, %f)\n", case, (*(complex_image_produit_image.crawdata+case-1)).real, (*(complex_image_produit_image.crawdata+case-1)).imag);

//--------------------fft inverse---------------------//
fft2(complex_image_produit_image, -1);

//--------------------conversion complexe2image---------------------//
bwimage_t  int_image = conversion_complexe2image(complex_image_produit_image);

/*
int *image = initialize_tab_i(complex_image1.width*complex_image1.height);
print_tab_i(image, 64);
print_tab_i(image, 64);

bwimage_t int_image = conversion_complexe2image(complex_image1.width, complex_image1.height, complex_image_produit_image.crawdata, image);
*/

//--------------------DumpImage---------------------//
E3ADumpImage("../../image/test3_corr2_image2_tester_output.png", &int_image);

release_tab_c(complex_image_produit_image.crawdata);
//release_tab_i(image); ?
return 0;

/*
int *image = initialize_tab_i(complex_image1.width*complex_image1.height);
print_tab_i(image, 64);
bwimage_t int_image = conversion_complexe2image(complex_image1.width, complex_image1.height, complex_image_produit_image, image);
print_tab_i(image, 64);


E3ADumpImage("../../image/output.png", &int_image);

release_tab_c(complex_image_produit_image);
release_tab_i(image);
return 0;
*/
```

```
}
```

Listing 26 – test4_deplace_image.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "projet.h"
#define case 55

int main(){
//--------------------image1---------------------//
bwimage_t image1;
error_e retval=E3A_OK;
retval=E3ALoadImage("../../image/test4_corr2_image1_motif_input.png", &image1);
Imagecomplexe complex_image1 = conversion_image2complexe(image1);
printf("(image1.width, image1.height) = (%d, %d)\n", image1.width, image1.height);
printf("(complex_image1.width, complex_image1.height) == (%d, %d)\n", complex_image1.width,
    complex_image1.height);
printf("%d -ieme element complexe de vector=(%f, %f)\n", case, complex_image1.crawdata[case
    -1].real, complex_image1.crawdata[case-1].imag);
fft2(complex_image1, 1); // fft

Imagecomplexe motif_deplace = deplacer_motif(complex_image1);

/*
gradient_image(complex_image1);
printf("%d -ieme element complexe de vector apres fft=(%f , %f)\n", case, complex_image1.
    crawdata[case-1].real, complex_image1.crawdata[case-1].imag);
//printf("3-ieme element complexe de vector apres fft -1 =(%f , %f)\n", complex_image1.crawdata
    [2].real, complex_image1.crawdata[2].imag);
*/

//--------------------fft inverse----------------------//
fft2(motif_deplace, -1);

//--------------------conversion complexe2image----------------------//
bwimage_t  int_image = conversion_complexe2image(motif_deplace);


E3ADumpImage("../../image/test4_corr2_image1_motif_output.png", &int_image);

//release_tab_c(complex_image_produit_image);
//release_tab_i(image);
return 0;
}
```

Listing 27 – main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "projet.h"
#define case 55

int main(){
//--------------------image1 : image a tester ----------------------//
bwimage_t image1;
error_e retval=E3A_OK;
retval=E3ALoadImage("../../image/main_tester_input.png", &image1);

Imagecomplexe complex_image1 = conversion_image2complexe(image1);

fft2(complex_image1, 1); // fft

//--------------------image2 : motif centre-------------------------//
bwimage_t image2;

retval=E3ALoadImage("../../image/main_motif_input.png", &image2);

Imagecomplexe complex_image2 = conversion_image2complexe(image2);

fft2(complex_image2, 1); // fft

//------------motif_deplace :  motif deplace au 4 coins ------------//
```

```
26 Imagecomplexe motif_deplace = deplacer_motif(complex_image2);
27
28 //------image produit image :  entre image1 et motif_deplace -------//
29 Imagecomplexe complex_image_produit_image=produit_image_version2(complex_image1, motif_deplace)
       ; // produit_image_version2 : meme principe que test_corr2.c ou il n'y a pas de terme de
       derivee croise
30
31 //--------------------fft inverse---------------------//
32 fft2(complex_image_produit_image, -1);
33
34 //----------------------conversion complexe2image-------------------------//
35 bwimage_t  int_image = conversion_complexe2image(complex_image_produit_image);
36
37
38 //--------------------DumpImage---------------------//
39 E3ADumpImage("../../image/main_tester_output.png", &int_image);
40
41
42 release_tab_c(complex_image_produit_image.crawdata); //release_tab_c : c'est le tab complexe
       utilise dans la structure nomee complex_image_produit_image
43
44 return 0;
45
46 }
```