Note technique

Date :29 octobre 2003

Mise à jour : 14 février 2006 (ajout de la descriptions des modules non décrits en 2003 + explications des changements apportés à la version 1.1 du BioEngine).

Service émetteur : R&D Auteur : Matthieu Perreira Da Silva

Produit : SDK DigiPass BioEngine

Sujet : Architecture et spécification de la version 1.1 du BioEngine DigiPass

Version : 1.1

Introduction

Les dernières performances de la pré-version 0.8b des algorithmes de matching ayant été jugées suffisantes, il a été décider de produire une version 1.0 de ces algorithmes.

Les méthodes de traitement et l'architecture globale de cette version seront celles de la version 0.8b. Les différences se situant au niveau de l'implémentation qui dans cette version 1.0a devra être faite de manière beaucoup plus propre et rigoureuse, afin de tenir compte des différentes contraintes que nous nous sommes imposées.

Ces contraintes sont :

- SDK développé pour une portabilité maximale (notamment au niveau des calculs utilisant des nombres à virgule).
- SDK entièrement écrit en C standard.
- Utilisation mémoire restreinte (dans la mesure du possible).
- Optimisation générale du code (mais pas d'optimisations pour une cible particulière pour l'instant)
- Format de gabarit compact.
- Utilisation réduite de fonctions trigonométriques ou mathématiques complexes (sin,cos,sqrt, log) et remplacement de ces fonctions par des tables dans la plupart des cas.
- SDK simple d'utilisation, comprenant un minimum de fonctions.
- Paramétrage limité au strict minimum

Cette version 1.0a est donc une ré-écriture de la version 0.8b afin de tenir compte des contraintes énoncées ci-dessus.

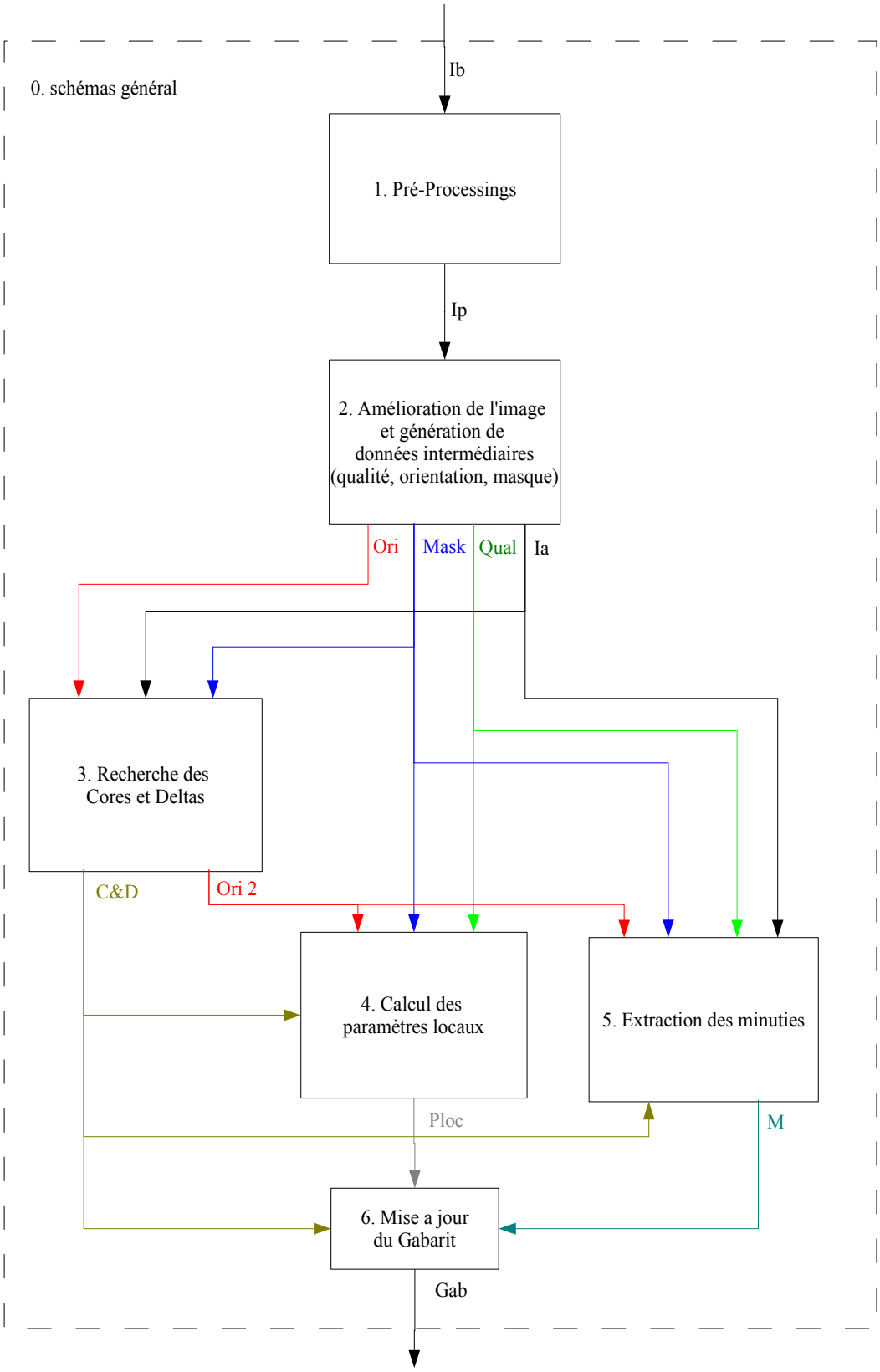
La suite de ce document va donc décrire l'architecture de l'algorithme qui sera implémenté dans la version 1.0a.

Courant 2004 et 2005 une version 1.1 de l'algorithme a été développés afin de permettre à celui-ci de pouvoir être compilé sur des plateformes différentes de la plateforme Win32 x86. Ont été rajoutées les fonctionnalités suivantes:

- Compilation sous Linux
- Possibilité de compiler l'algorithme pour qu'il effectue tous ses calculs en virgule fixe (nécessaire aux tests de l'algorithme sur processeur ARM9).
- Optimisations de vitesse et modification de l'algorithme (mise à l'échelle de l'image d'entrée, modifications de certains paramètres et fonctions, etc.) pour que la vitesse de l'algorithme soit réglable (jusqu'à 3 fois plus rapide que dans sa version « normale »).
- Optimisation des performances générales de l'algorithme (vitesse et taux de FAR/FRR).

Ce document a donc été modifié pour présenter l'architecture de la version 1.1 de l'algorithme.

I. Architecture générale de l'algorithme de génération de gabarits.

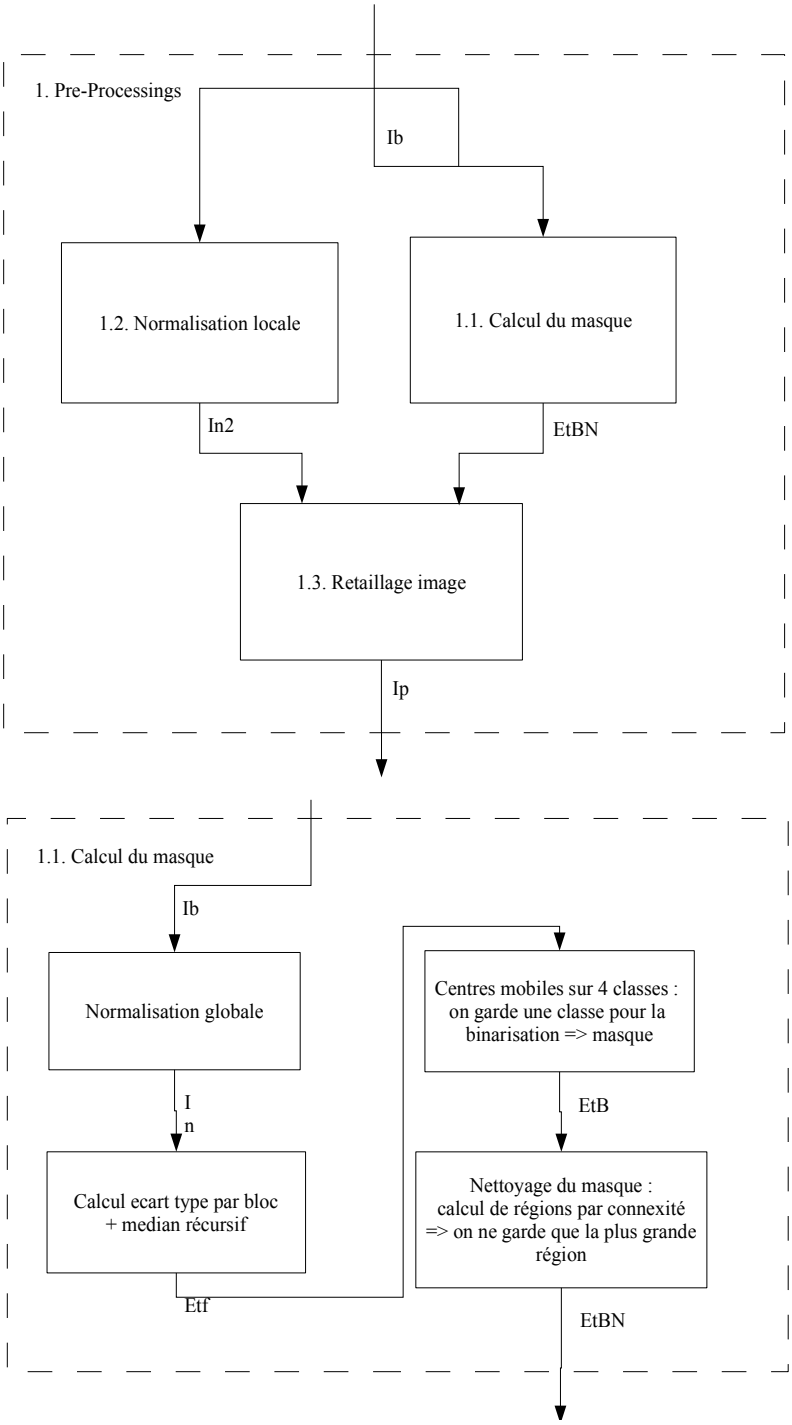


La figure 1 décrit l'architecture générale de l'algorithme. On y trouve 6 blocs fonctionnels dont les

fonctions sont les suivantes :

1. **Pré-processings :** Ce bloc prend en entrée l'image brute *(I_b)* issue d'un capteur d'empreintes digitales et lui applique différents pré-traitements. Ces pré-traitements consistent en une normalisation ainsi qu'en une première détermination de la zone utile de l'image (zone contenant l'image d'empreinte). A partir de cette zone utile, l'image est retaillé afin de limiter les temps de calcul dans les phases ultérieures.

   En sortie de ce bloc, on obtient une image normalisée et retaillé *(I_p)*.



1.1 Calcul du masque

La première étape du traitement consiste en une normalisation globale : la moyenne globale de tous les pixels de l'image ainsi que leur écart type sont fixés à une valeur prédéterminée (typiquement : moyenne=127, écart-type=50).

Pour ce faire, on effectue donc les opérations suivante :

- Calculer $Mean = \frac{1}{N}\sum_{x=0}^{N-1} I_b(x)$ avec N=LxH le nombre de pixels total de l'image I_n
- Calculer $StdDev = \sqrt{\frac{1}{N}\sum_{x=0}^{N-1} (I_b(x))^2 - Mean^2}$
- Calculer la nouvelle image normalisée I_n, selon la formule suivante:

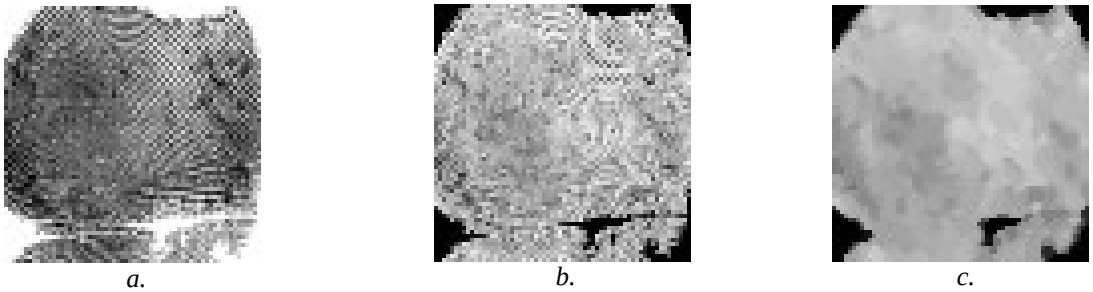
$$I_n(x) = TargetMean + \frac{(I_b(x) - Mean) * TargetStdDev}{StdDev}$$

Ib : image d'origine (taille:512x512)

In : Image normalisée globalement (taille:512x512)

La seconde étape est le calcul des moyennes et écart types locaux de l'image, sur différents blocs de taille T_1xT_1 (avec T_1=8 habituellement). Ces blocs de taille T_1xT_1 ne se recoupent pas. On a donc par exemple 16x32=512 blocs sur une image 128x256. Le calcul est le même que les étapes 1 et 2 de la normalisation globale, mais en effectuant le traitement bloc par bloc (calcul sur T_1xT_1 pixels au lieu de LxH pixels).

Les moyennes et écarts types de chaque blocs sont stockés dans les images E_m et E_t de taille $N_1=L_1xH_1$, avec $L_1=l/T_1$ et $H_1=H/T_1$.
On modifie cette image par un filtre médian récursif de taille T_2 (avec T_2=3 typiquement) afin d'obtenir l'image filtrée E_{tf}.



a. *b.* *c.*

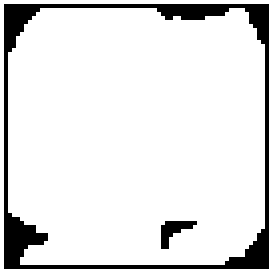
a. Image E_m des moyennes de chaque bloc, b. Image E_t des écarts types de chaque bloc, c. Image E_{tf} des écarts types filtrés de chaque bloc. Chaque image a une taille de 64x64

On applique ensuite un algorithme de centres mobiles afin de réduire la dynamique de notre image d'écarts types E_t de 256 niveaux à N_{Cm} niveau (N_{Cm}=4 typiquement). On obtient l'image E_{tfcm}.

E_{tfcm}. Image des écarts types de chaque bloc filtrée sur laquelle un algorithme de centre mobile à 4 classes a été appliqué (taille:64x64).

L'image E_{tfcm} est ensuite binarisée en fonction des niveaux calculés par l'algorithme des centres mobiles afin d'obtenir le masque E_{tB}. On a donc :

$$E_{tB}(x) = \begin{matrix} 0\, si\, E_{tfcm}(x) < 1 \\ 1\, si\, E_{tfcm}(x) \geq 1 \end{matrix}$$



E_{tb}. Masque binarisé (taille:64x64)

Le masque binaire est finalement nettoyé par l'algorithme suivant, permettant de supprimer tous les pixels parasites et ne garder qu'une seule grande région connexe :

Pour chaque pixel de l'image de valeur 1 :
    s'il existe un pixel connexe de valeur 1, ajouter le pixel à la région du pixel connexe
    sinon créer une nouvelle région
Fin pour
Calculer la taille de chaque région

Pour chaque région :
Si taille = taille de la région la plus grande, laisser tous les pixels de la region à 1
Sinon mettre tous les pixels de la région à 0.
Fin pour

L'image I_p ainsi traitée par cet algorithme ne représente ensuite plus qu'une seule région formée de pixels connexes. C'est notre masque final E_{tBN}.

E_{tbn}. Masque finale nettoyé (taille:64x64)

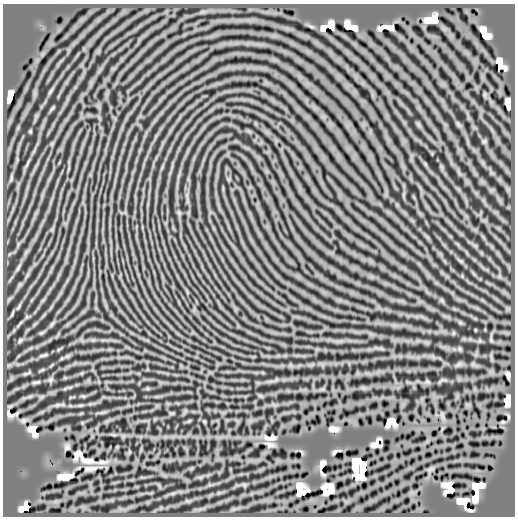
1.1 Normalisation locales

La normalisation locale utilise l'algorithme suivant :

- calcul de la moyenne et de l'écart type de chaque bloc, comme décrit plus haut dans la partie calcul du masque
- interpolation des moyennes et écarts type afin d'obtenir une valeur de moyenne et d'écart type pour chaque point de l'image d'origine.
- Modification de la valeur de chaque pixel de l'image afin d'obtenir les bonnes valeurs de moyenne et d'écart type en chaque point :

$$I_{n2}(x) = TargetMean + \frac{(I_b(x) - Mean(x)) * TargetStdDev}{StdDev(x)}$$

L'image I_{n2} obtenue présente maintenant des moyennes et écarts type locaux (c'est à dire calculés pour un voisinage donné) dont les valeurs sont TargetMean et TagetStdDev.



I_{n2}. Image normalisée localement (taille:512x512)

1.1 Retaillage de l'image

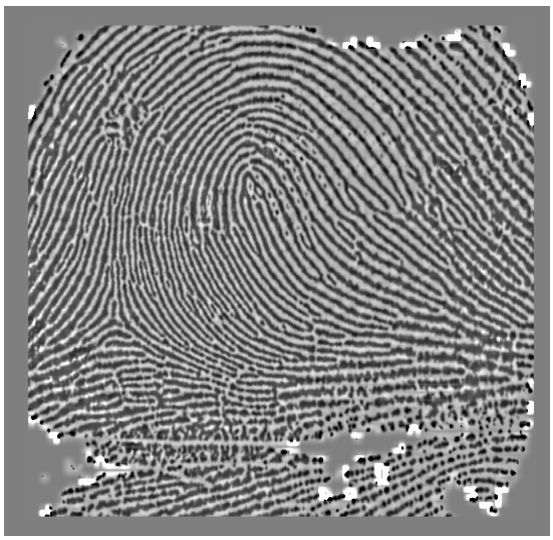
Le retaillage de l'image satisfait deux objectifs :

- réduire la taille de l'image afin de diminuer les temps de traitement lors de la génération du gabarit. Cette diminution de la taille de l'image permet également de limiter les risques de fausse détection de minuties à des endroits de l'image ne faisant pas partie de l'empreinte digitale elle même.
- Générer une image dont la taille sera un multiple de la taille des blocs (se chevauchant) utilisés pour l'amélioration de l'image, afin de faciliter le traitement de ces blocs.

Préalablement à l'opération de retaillage elle même, on calcule (à partir du masque E_{tBN}) la nouvelle taille de l'image ainsi que la position de l'ancienne image dans notre image retaillée afin que :

- on laisse une marge (correspondant a la largeur de chevauchement entre les blocs) de chaque coté de l'image afin que les artefacts produits par le filtrage par FFT ne soient pas perceptible sur les bords de l'empreinte.
- La taille total de l'image permettent de pouvoir la découper en un nombre entier de blocs de taille $T_{bFFT}xT_{bFFT}$ se chevauchant de T_{chev}.

L'image ainsi retaillée pourra ensuite être traitée par l'algorithme d'amélioration d'image.



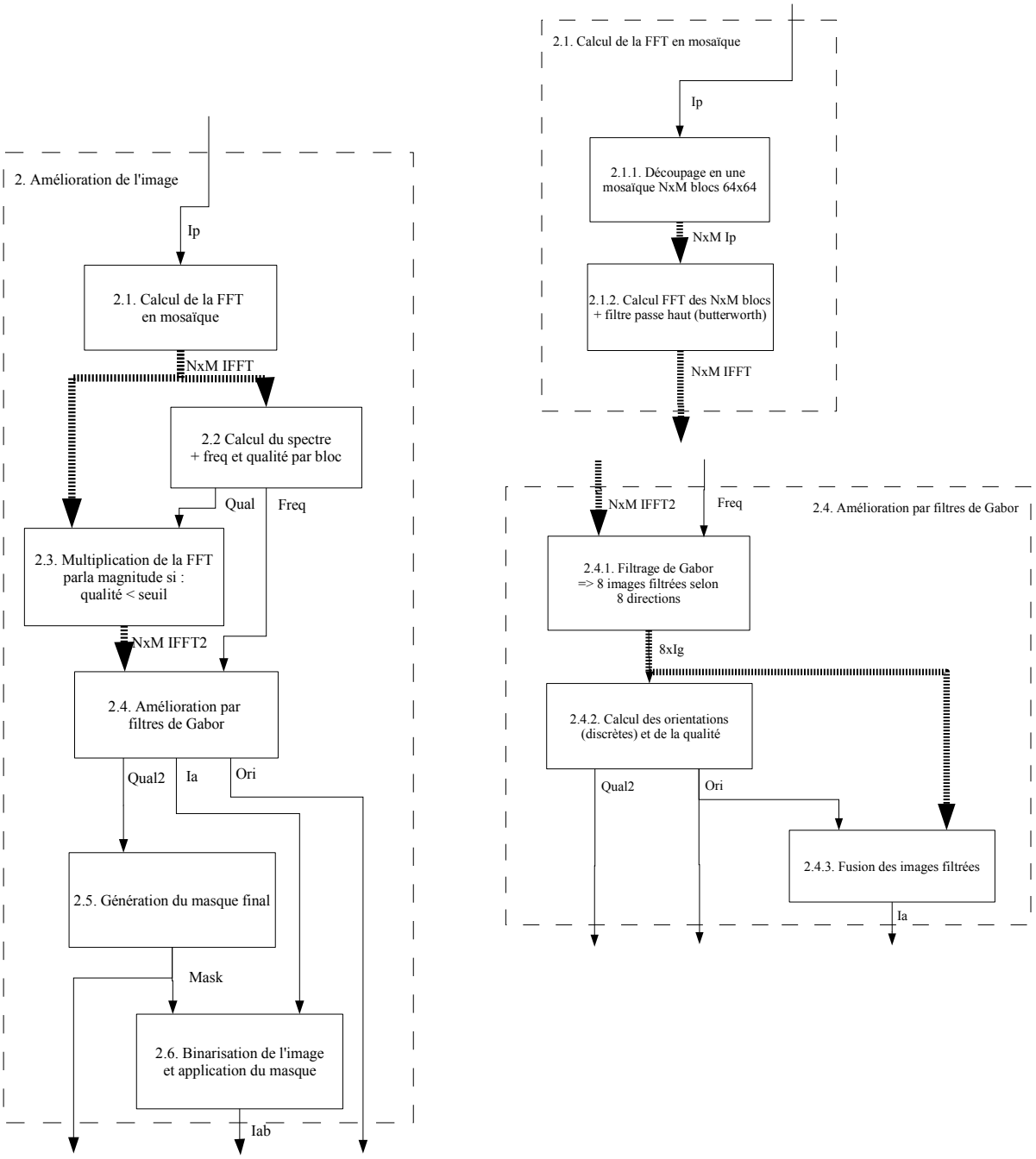
I_p. Image après préprocessing et recadrage (taille:544x544)

1. Amélioration de l'image

C'est dans ce bloc qu'est effectué le gros du traitement de l'image visant à améliorer sa qualité. On obtient en sortie une image améliorée binarisée *(Iab)*. Cette étape génère également 2 autres données :

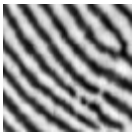
- Une carte des orientations locales de l'image *(Ori)*. Elle sera utile pour déterminer la position des cores et deltas dans l'image ainsi que l'orientation des minuties et certains paramètres locaux de l'image.
- Une carte de qualité de l'image *(Qual)*, indiquant la plus ou moins bonne qualité locale de l'image. Cette carte sera utilisée pour générer certains paramètres locaux ainsi que pour indiquer la qualité des différentes minuties extraites en 5. Elle sert aussi de masque indiquant les zone de l'image représentant réellement des zones d'empreinte (partie utile). Ce masque est utilisé par tous les blocs de traitement restant (3,4 et 5) afin de déterminer sur quelles

parties de l'image effectuer (ou ne pas effectuer) les traitements.



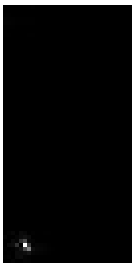
2.1 Calcul de la FFT en mosaïque

Ce premier bloc de traitement crée tout d'abord un tableau de taille NxM de blocs (sous parties de l'image de départ) de taille T_{enh}xT_{enh} (avec T_{enh} = 64 typiquement). Ces blocs se chevauchent de $T_{chev.}$



Exemple de bloc de taille 64x64

On applique ensuite un algorithme de FFT sur chacun de ces blocs afin d'obtenir leur transformée de Fourier. On obtient alors un tableau de NxM blocs IFFT.

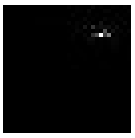


I_{FFT}. Exemple de FFT de bloc 64x64 (ici le bloc fait 32x64 car on ne représente que la magnitude de la FFT)

2.2 Calcul des fréquences moyenne (et qualité de cette estimation) par bloc

Afin de calculer la fréquence moyenne de chaque bloc, il est nécessaire de calculer la réponse spectrale de chaque bloc. C'est ce que l'on effectue dans une première phase de ce bloc de traitement.

On applique tout d'abord un algorithme permettant de calculer le spectre du bloc FFT (la norme de ses parties imaginaires et réelles) en coordonnées polaires. L'utilisation de coordonnées polaires est commode car elle permet de distinguer facilement les composantes du bloc représentant la fréquence de celles représentant l'orientation.
Après ce traitement on obtient un tableau de NxM blocs Ispec. Pour chaque bloc, une ligne représente une fréquence déterminée et une colonne, une orientation déterminée.



Ispec. Exemple de spectre d'un bloc (taille = 32x32)

On applique ensuite un algorithme de détection de la fréquence la plus importante dans chaque bloc. Pour évaluer cette fréquence, chaque bloc est découpé en N_{band} (N_{band}=8) bandes de fréquence. On choisira ensuite la la fréquence la plus importante dans la bande de fréquence de plus haute énergie Cet algorithme évalue également la pertinence de sa mesure. Pour certains blocs dont la mesure de fréquence est jugée non pertinente, une nouvelle évaluation est réalisée en interpolant la valeur à partir de celle de ses voisins dans le tableau de blocs.

L'algorithme est le suivant :

Pour tous les blocs de spectre en coordonnées polaires
    Pour chaque bande de fréquence
        calculer l'énergie E de la bandes
        calculer l'écart type Et_E des énergies autour de l'orientation moyenne
        calculer l'importance I_{band} de la bande $I_{band}=E/Et_E$
    Fin pour
    Chercher l'index I_{bMax} de la bande dont I_{band} est le plus important

$$I_{bMean} = \frac{1}{N_{band}-1} \sum_{x=0,\, x \neq I_{bMax}}^{N_{band}-1} Iband(x)$$

Calculer l'intensité moyenne des autres bandes :

Calculer la fréquence moyenne F_{moy} de la bande I_{bMax} ayant la plus forte énergie
Calculer la qualité de l'estimation : *Qual* = $I_{band}(I_{max}) / I_{bMean}$
Fin pour

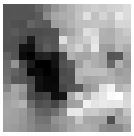
NbBlocsCorrects = 0
frequenceMoyenne = 0
Faire
Pour tous les blocs de spectre en coordonnées polaires
Si *Qual* du bloc < *seuil*
marquer le bloc comme mauvais
Sinon
frequenceMoyenne = *frequenceMoyenne* + *freq* du bloc
NbBlocsCorrects = *NbBlocsCorrects* + 1
Fin si
Fin pour
Baisser seuil
Tant que *NbBlocsCorrects* < *seuilNbBlocs*

frequenceMoyenne = *frequenceMoyenne* / *NbBlocsCorrects*

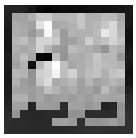
Pour tous les blocs marqués comme mauvais
Si nombre de voisins maqués comme bons > *seuil2*
I**nterpoler** la valeur de fréquence *freq* à partir de la fréquence des voisins marqués comme bons
I**nterpoler** la valeur de qualité *qual* à partir de la qualité des voisins marqués comme bons.
Marquer le bloc comme bon
fin si
Fin pour

Pour tous les blocs marqués comme mauvais
freq = *frequenceMoyenne*
Fin pour

Cet algorithme permet d'obtenir une bonne estimation de la fréquence moyenne des lignes d'empreintes pour chaque blocs, ainsi qu'un indice de confiance correspondant à la fidélité de cette estimation.



Freq. Carte des fréquences de l'image (taille:16x16)



Qual. Estimation de la qualité des fréquences de Freq (taille:16x16)

2.3 **Multiplication de la FFT par sa magnitude**

Ce traitement est un traitement conditionnel qui n'est effectué que sur les blocs dont la qualité

d'estimation de la fréquence est faible. Il permet d'améliorer la clarté des lignes d'empreintes dans ces régions. Cependant, comme ce traitement supprime également certains détails de l'image, il n'est pas utilisé sur les blocs de bonne qualité.

Le principe du traitement est de multiplier chaque valeur complexe de la FFT de chaque bloc par une puissance réelle de sa magnitude :

$$Magnitude(x) = (ReFFT(x)^2 * ImFFT(x)^2)^{(n)}$$ avec n=0.25 typiquement

ReFFT(x) = ReFFt(x) * Magnitude(x)
ImFFT(x) = ImFFT(x) * Magnitude(x)

Une normalisation est ensuite effectuée afin de ne pas générer de données saturées.
Chaque valeur complexe de la FFT est divisée par le Max de magnitudes du bloc.

2.4 **Amélioration par filtres de Gabor**

Les filtres de gabor sont des filtres sélectifs en orientation et fréquence. Ils permettent de faire ressortir des lignes espacés selon un certaine fréquence et orientées dans une certaine direction. Le principe de notre algorithme de filtrage est donc d'utiliser un banc de filtres de gabor, dont la fréquence est déterminée par les fréquences Freq définies en 2.2.
On utilise $N_{or}i$ (Nori=8) orientations différentes pour définir les orientations du banc de filtres.
Les blocs sont filtrés pas les N_{ori} filtres. N_{ori} images différentes sont reconstituées à partir des NxMxNori blocs. Un processus de sélection d'orientation basé sur l'importance de la réponse des filtres ainsi que sur la continuité des orientations permet de définir pour chaque pixel de l'image finale I_a, le bon pixel à écrire. Ce processus génère également une carte des orientations Ori ainsi qu'une autre carte de la qualité de l'image finale Qual2.

On obtient au final une image I_a dont les lignes ont été mises en valeur par des filtres sélectifs en orientation et fréquence. Il est important de noter que le choix de l'orientation est repoussé le plus loin possible dans le traitement afin de posséder des information de haute qualité pour la sélection des orientations. Cette méthode permet une amélioration efficace des images, même très bruitées.

2.4.1 **Application des filtres de Gabor**

Cette partie de traitement filtre chacun des NxM blocs avec Nori filtres de Gabor. Ces Noris filtres couvrent 180°, pour 8 filtres on aura les orientation suivantes : 0°, 22.5°, 45°, ..., 157.5°.
La fréquence des filtres appliqués à chaque bloc est définie à partir de la carte des fréquences (Freq) calculée en 2.2.
Le filtrage est effectué par transformée de Fourier rapide, multiplication dans le domaine fréquentiel puis transformée de Fourier inverse. Les filtres sont calculés directement dans le domaine fréquentiel, en temps réel.
Chaque bloc traité est ensuite incorporé dans une des NorixIg images de sortie. Les blocs se chevauchant il faut les multiplier par un masque avant de les ajouter à l'image de destination afin de «mélanger » correctement les images dans les zones de chevauchement.

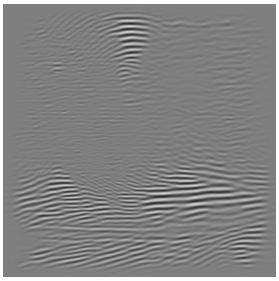


Image filtrée (ori=0°, taille:544x544)

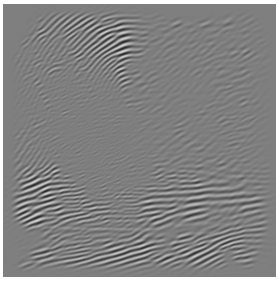


Image filtrée (ori=22.5°)

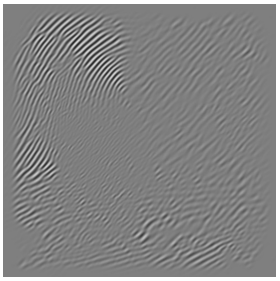


Image filtrée (ori=45°)

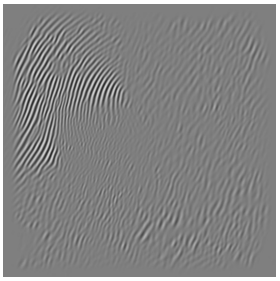


Image filtrée (ori=67.5°)

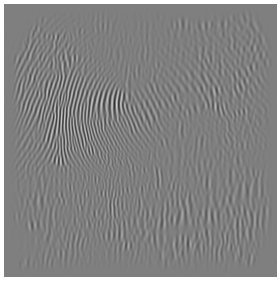


Image filtrée (ori=90°)

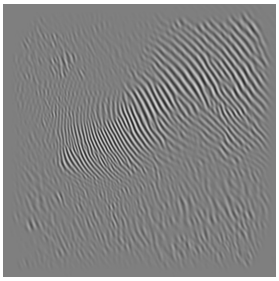


Image filtrée (ori=112.5°)

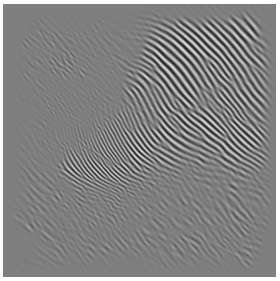


Image filtrée (ori=135°)

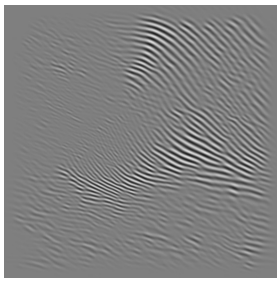
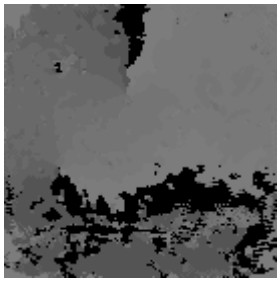


Image filtrée (ori=15 7.5°)

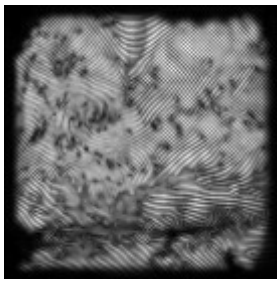
2.4.2 **Détermination des orientations et de la qualité à partir des images filtrées**

La détermination des orientations et de la qualité s'effectue en plusieurs étapes :

1. Première estimation : Cette estimation est effectuée en calculant pour $N_a x M_a$ blocs de taille T_{blocA} (T_{blocA} = 4 typiquement) l'intensité moyenne de chaque image Ig filtrée. On ecrit dans une image Ori l'orientation du bloc ayant la plus forte intensité, et dans Qual2 l'intensité de ce bloc.



Ori. Première estimation des orientations de l'image (taille:136x136)

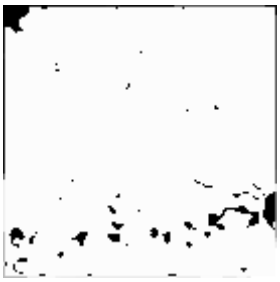


Qual2. Première estimation de la qualité des orientations (taille:136x136)

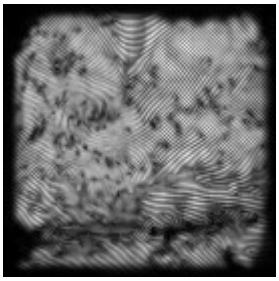
2. La carte des orientations Ori contenant de nombreuses discontinuités, on applique un filtre médian afin de réduire celles-ci et obtenir une représentation plus homogène.

Ori. Deuxième estimation des orientations (après filtre médian) (taille=136x136)

3. On recherche ensuite dans l'image Ori ainsi filtrée toutes les régions formées par des pixels connexes ayant la même orientation. On va ensuite filtrer ces régions en fonction de leur taille et de leur orientation afin d'obtenir des transitions douces lors des changement d'orientation dans l'image. Pour cela, on remplace les régions qui sont trop différentes (différence supérieure à un seuil) de leur voisins, par la valeur d'orientation la plus probable, calculée en fonction de l'orientation et de la qualité des différents pixels voisins au pixel de la région traitée. La carte de qualité Qual2 est mise à jour en fonction de l'intensité des pixels remplacés.



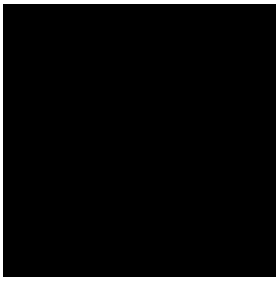
En noir, les pixels à réévaluer (taille:136x136)



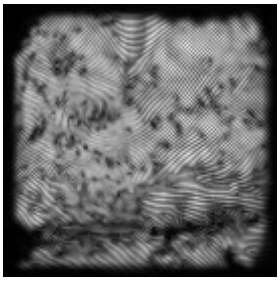
Qual2. Nouvelle estimation de la qualité des orientations (taille:136x136)

Ori. Orientations corrigées (taille:136x136)

4. On applique ensuite le même traitement mais pixel par pixel afin de réduire les problèmes dûs à des changements brusques d'orientation. Pour effectuer ce traitement, on recherche tous les pixels dont la valeur est trop différentes de ses voisins. On stocke ces pixels dans une image I_{pixel} sur laquelle on va appliquer une dilatation afin d'agrandir un peu la zone à traiter. On applique ensuite le même algorithme de remplacement des orientations qu'en 3 pour les pixels marqués comme « à traiter ». La carte de qualité Qual2 est mise à jour en fonction de l'intensité des pixels remplacés.



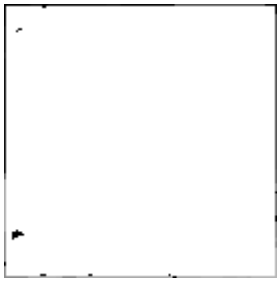
Pixels à modifier (ici aucun pixel à modifier, donc l'image est toute noir) (taille:136x136)



Qual2. Estimation de la qualité des orientations suite aux pixels modifiés (ici aucun) (taille:136x136

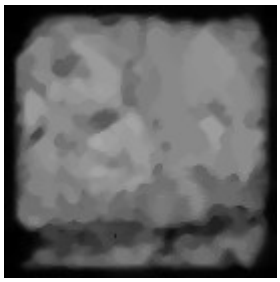
Ori. Estimation finale des orientations (taille:136x136)

5. On recherche une dernière fois les régions qui sont trop différentes de leurs voisines afin de déterminer si les traitements effectués avant ont eu un effet bénéfique. Si ce n'est pas le cas, on marque ces régions comme mauvaises dans la carte de qualité.



Dernière estimation des région dont la qualité est mauvaise

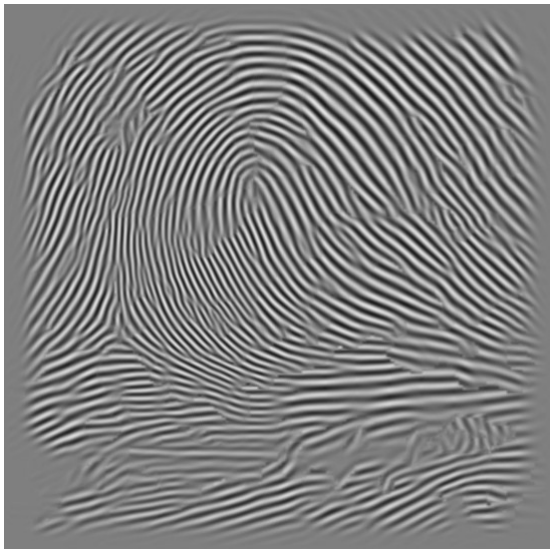
6. Un filtre médian est appliqué à la carte de qualité.



Qual2. carte de qualité des orientation, version finale (après filtre médian et corrections de l'étape précédente) (taille:136x136)

2.4.1 **Fusion des images filtrées en une image améliorée**

On sélectionne les pixels à écrire dans l'image fusionnée finale en fonction de la carte d'orientation calculé en 2.4.2. En fonction de cette carte on choisit chaque pixel dans une des N_{ori} images filtrée. Afin de rendre les transitions plus douces, une interpolation entre les différentes orientations est réalisée au frontières de chaque bloc (l'orientation étant définie bloc par bloc).



Ia. Image amélioré : fusion à partir de la carte des orientations des différentes images filtrée par filtre de Gabor (taille:544x544)

2.1 **Génération du masque final**

Le masque final est seuillé afin de séparer les parties de l'image correspondant au fond et les parties de l'image correspondant à l'empreinte.

L'algorithme est le suivant:

Calculer la qualité maximum de l'image Q_{max}
Pour tous les indices de qualité
    Si $Qual2 < seuil3\% * Q_{max}$
        $Mask = 0$
    Sinon
        $Mask = Qual2$
    Fin Si
Fin pour

On obtient donc un masque Mask qui contient la qualité de l'empreinte dans les zone d'empreinte et 0 dans les zones correspondant au fond.

2.6 Binarisation de l'image et application du masque

Ce bloc permet de générer une image d'empreinte binarisée, directement exploitable par le bloc effectuant la squelettisation de l'image et la recherche de minuties dans le squelette ainsi obtenu.

Les traitements effectués dans ce bloc sont les suivants:

- Binarisation de l'image : L'image améliorée Ia générée par le Bloc 2.4.3 est seuillée par un algorithme basique. Si le niveau de gris du pixel traité est supérieur au seuil (par défaut 127) alors le pixel sera écrit avec la valeur 255. Si le niveau de gris est inférieur au seuil ,alors le pixel sera écrit avec la valeur 0.

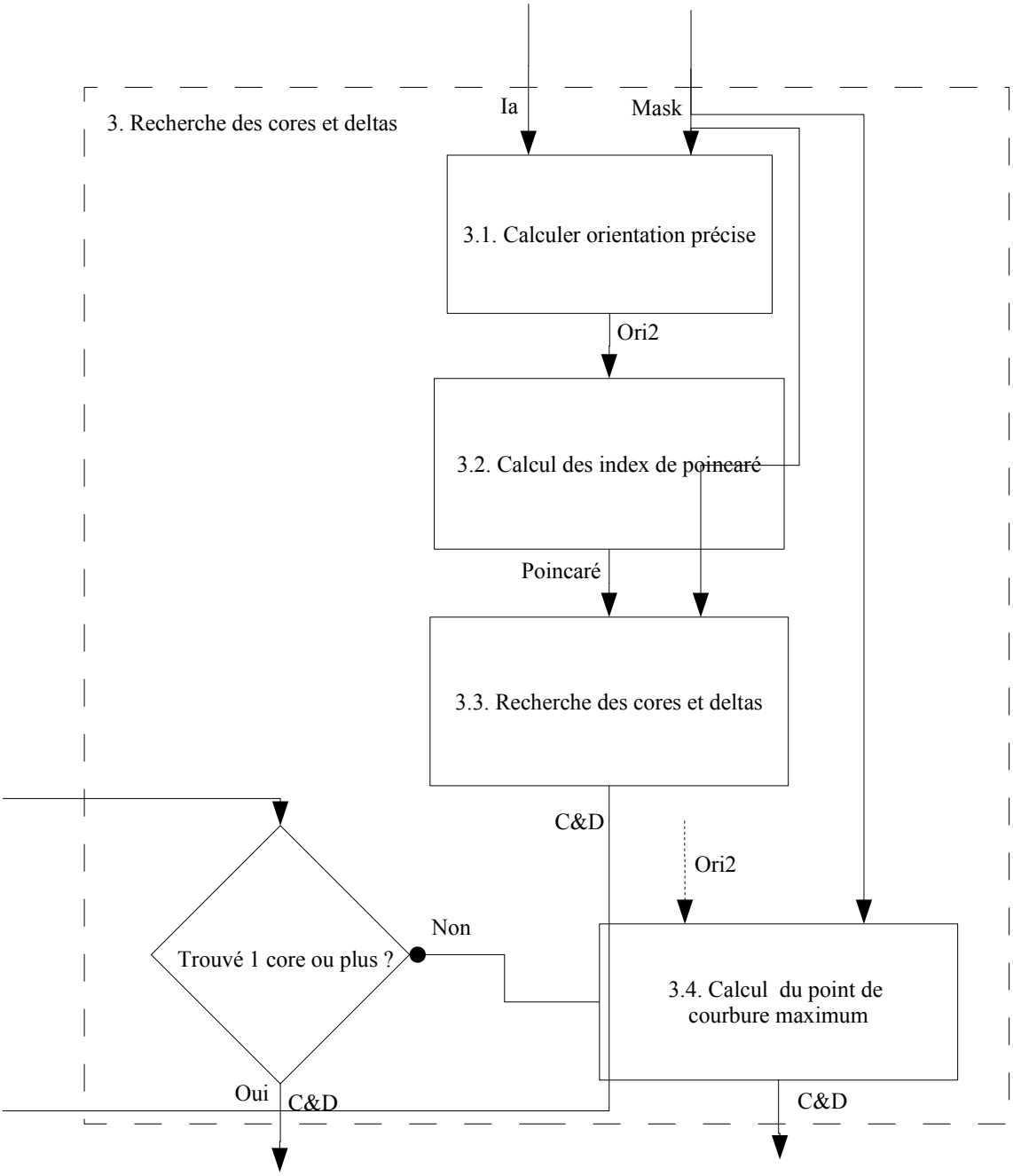
- Masquage de l'image : Un autre algorithme très simple est appliqué à l'image. Pour chaque point de l'image d'empreinte binarisée à traitée, on consulte la valeur de la carte de qualité (masque) correspondante. Si la qualité a pour valeur zéro alors le pixel de l'image d'empreinte est mis à 255. Cette opération permet de supprimer de l'image les zone ne contenant pas de données exploitables.

- Filtrage de l'image binarisée et masque à l'aide d'un opérateur morphologique : le filtre médian récursif. Celui-ci permet de « boucher » les éventuels trous qui auraient pu persister dans les lignes papillaires après amélioration de l'image.

Ib : image binarisée et masquée + médian recursif (taille:544x544)

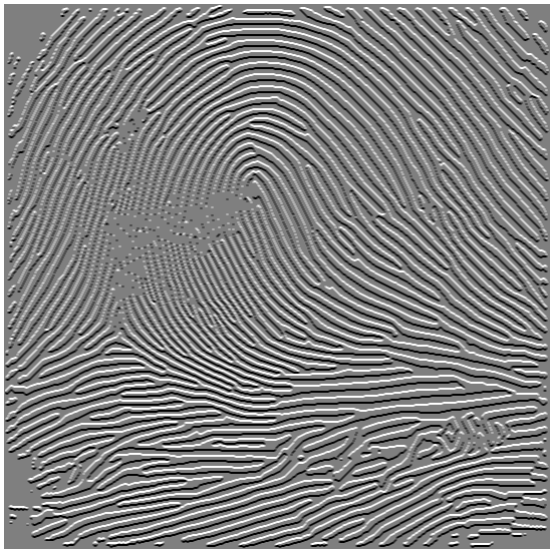
3. Recherche des cores et deltas

Ce bloc recherche des zones ayant des caractéristiques précises dans l'image (changements brusques de courbure). La position de ces points *(C&D)* sera utilisée comme référence pour la génération des paramètres locaux (4) et des minuties (5). Elle sera également stockée dans le gabarit afin d'être utilisée lors du matching. Lors de cette étape on génère également une nouvelle carte des orientations (plus précise) qui sera utilisée pour définir l'orientation des minuties.



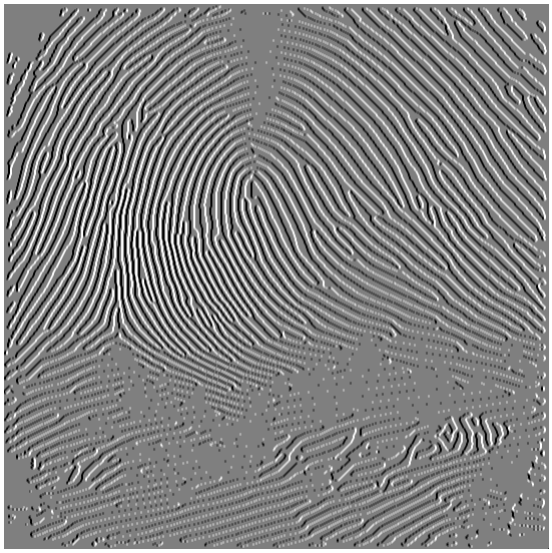
3.1 **Calcul de l'orientation précises**

Les orientations sont calculées à partir du gradient de l'image. On utilise donc des filtres de Sobel afin de calculer les composantes en X et Y du gradient de l'image améliorée I_a. Ces composantes sont stockées dans deux images : G_x et G_y.



G_x : image représentant la composante horizontale du gradient de l'image I_a

(Taille 544x544)



G_y : image représentant la composante verticale du gradient de l'image I_a

(Taille 544x544)

A partir du gradient de l'image on calcule les orientations de l'image par bloc en utilisant la méthode des moindres carrés. Pour chaque bloc de l'image de taille $T_{blocOri}$, on calcule l'orientation en utilisant la formule suivante :

$$Vx = \sum_{u=-Tblocori/2}^{Tblocori/2} \sum_{v=-Tblocori/2}^{Tblocori/2} \left(2Gx(u,v)Gy(u,v)\right)$$

$$Vy = \sum_{u=-Tblocori/2}^{Tblocori/2} \sum_{v=-Tblocori/2}^{Tblocori/2} Gx(u,v)^2 - Gy(u,v)^2$$

$$OriBloc = \frac{1}{2}\arctan\left(\frac{Vx}{Vy}\right)$$

On obtient ainsi une carte des orientations Ori2 de taille L/TblocOri x H/TblocOri (avec L et H la taille de l'image améliorée Ia). Cette orientation est calculée sur des blocs de taille TblocOri espacés de TstepOri (et donc se chevauchant).

On aura par exemple :

si $Taille_{Ia}$ = 544x544, $T_{blocOri}$=16 et $T_{stepOri}$=4 : $Taille_{Ori2}$ = 544*4/16 = 136x136

Ori2 : carte des orientations de l'image avec $T_{blocori}$ = 16 (taille:136x136)

Cette image étant trop petite pour pouvoir être exploitée facilement dans les étapes suivantes elle est donc agrandie pour obtenir une nouvelle image Ori2 de même taille que Ia.

Carte des orientation Ori2 agrandie à la taille de I_a (544x544)

3.2 **Calcul des indexes de Poincaré**

L'étape suivante est le calcul des indexes de Poincaré à partir de la carte d'orientation Ori2. Ces indexes vont permettre de calculer exactement le nombre et la position des cores et deltas.

L'indexe de Poincaré de chaque point de l'image Ori est calculé à partir de son 8 voisinage.

d2	d3	d4
d1	Ori2(i,j)	d5
d0	d7	d6

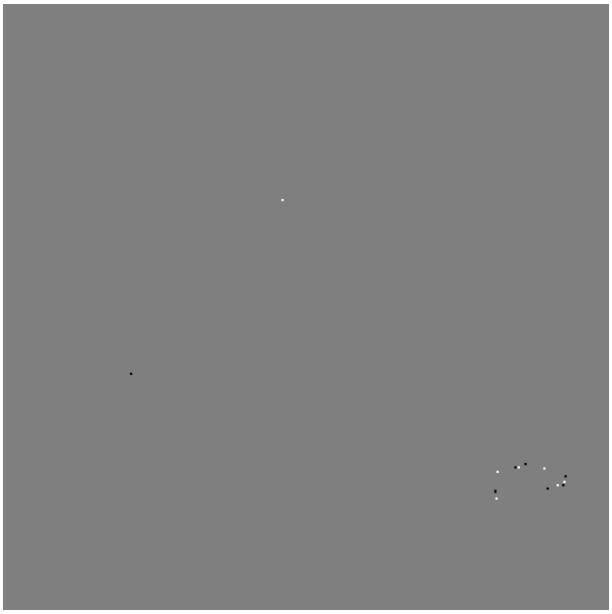
La formule de calcul de l'indexe de Poincaré est la suivante :

$$P(i,j)=\frac{1}{2}\pi\sum_{k=0...7}D(k)$$

$$D(k)=\begin{cases} angle(k), si\,|angle|(k)<\frac{\pi}{2} \\ \pi+angle(k), si\,angle(k)\leq\frac{-\pi}{2} \\ \pi-angle(k), sinon \end{cases}$$

$$avec\quad angle=d_k-d_{(k+1)mod\,8}$$

En appliquant cette formule à tous les pixels de la carte Ori2, on obtient une image contenant les cores (P(i,j)=180°) et deltas P(i,j) = -180°). Cette image contient également des fausses détections qu'il faudra filtrer.



Poincaré : Cores (blanc), Deltas (noir) ainsi que points parasites (coin bas droite) (taille:544x544)

3.3 **Recherche des cores et deltas**

Les cores et deltas sont ensuite détectés dans l'image Poincaré. Les points dont l'indexe est égale à 180° sont détectés comme des cores et ceux dont l'indexe est égale à -180° comme des deltas. On élimine les points parasites en utilisant les règles suivantes:

- Si plusieurs cores/deltas connexes sont détectés alors on ne retient qu'un seul core/delta donc la position est le centre de gravité de l'ensemble de cores/deltas.
- Si un core/delta est détecté trop proche (au delà d'un seuil paramétrable) d'un autre core/delta alors celui-ci n'est pas pris en compte car il n'est pas normal d'avoir des cores / deltas proches.

Ces deux règles simples permettent d'éliminer un maximum de points parasites.

3.4 **Calcul du point de courbure maximum**

Si le calcul des indexes de Poincaré ne permet pas de trouver un core (cas des empreintes de type Arche simple), on utilise alors une autre méthode permettant de calculer le point de courbure maximum de l'image.

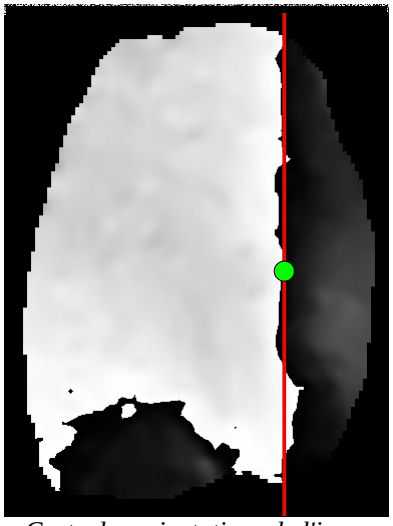
Dans le cas des images en arche, ce point se trouve toujours sur l'axe de symétrie verticale de l'image. La première étape du traitement est donc de recherche cet axe.

Recherche de l'axe de symétrie vertical :

Pour les images de type arche simple, l'axe de symétrie passe par la zone ou l'orientation des lignes d'empreintes est égale à zéro. Les orientations ayant des valeurs comprises entre 0 et 180° (mappée entre 0 et 255 pour des raisons d'implémentation), l'objectif de l'algorithme de recherche de l'axe de symétrie est de trouver l'axe vertical autour duquel les valeurs d'orientation passent brutalement d'une valeur proche de 255 (180°) à une valeur proche de zéro (0°).

.

Imagé améliorée d'une empreinte de type arche simple (taille:384x512)



Carte des orientations de l'image (taille:384x512). En rouge, l'axe vertical de symétrie. En vert le point de courbure maximum.

Pour déterminer la position de cette axe vertical on applique les calculs suivants :

$$MeanY(x) = \sum_{y=0}^{imgWidth} Orientation(x, y)$$

$$VertAxis = max(GradientMeanY) \quad avec$$

$$GradientMeanY(x) = MeanY(x) - MeanY(x+1)$$

Recherche du point de courbure maximum :

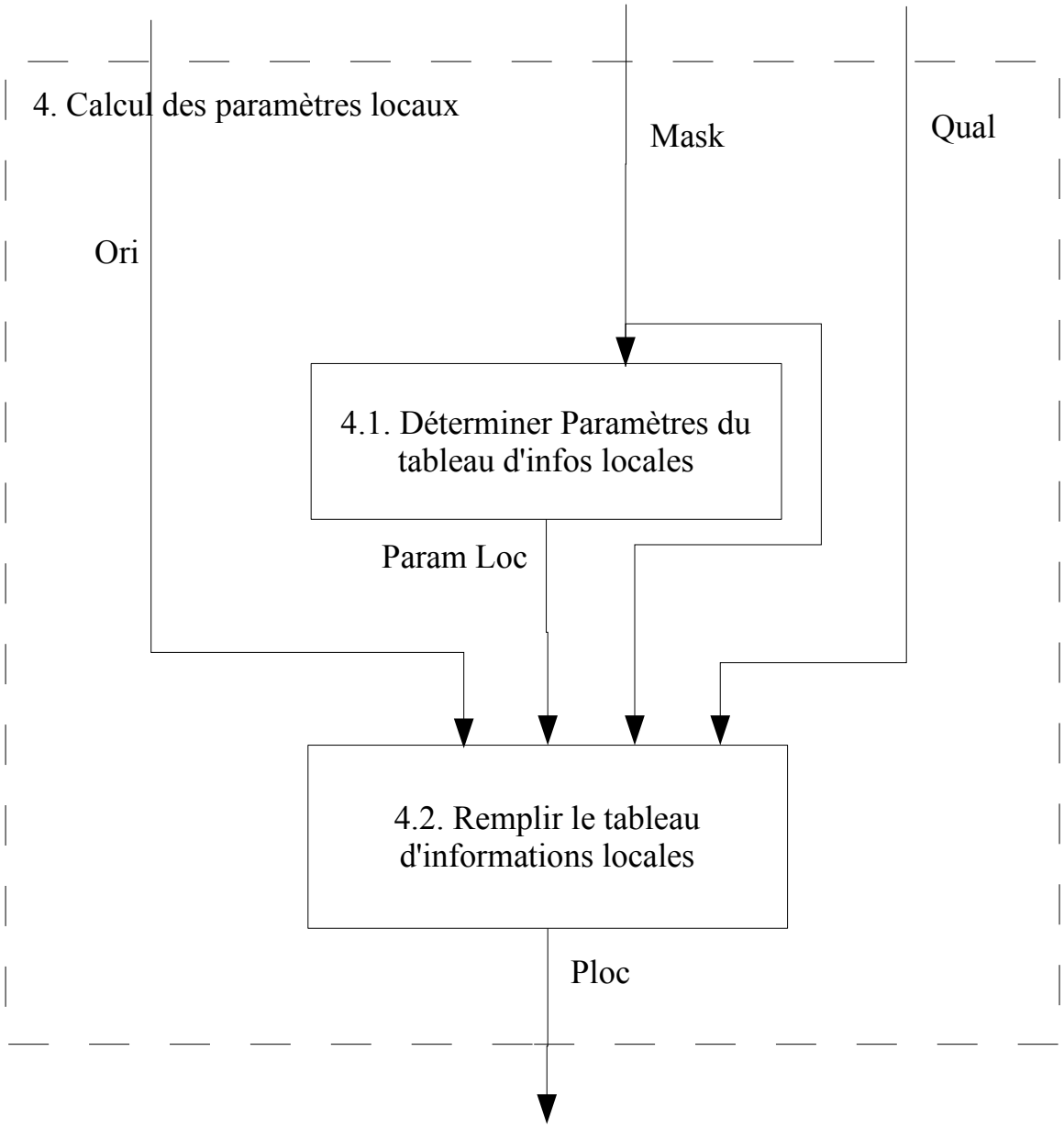
Il ne reste ensuite plus qu'a trouver le point de courbure maximum, situé sur cet axe. La méthode de calcul de ce point est assez archaïque et pourrait être améliorée, mais a été jugé pour l'instant suffisante étant donnée le rôle peu important des cores lors de l'authentification (ils aident par contre à accélérer l'identification).

Le principe de ce calcul est d'additionner pour chaque ligne de l'image les orientations située dans une zone autour de l'axe de symétrie. Les angles proches de 180° (255 en valeur d'orientation) sont transformés en angles proches de 0°, puisque nous ne nous intéressons qu'à l'importance de la pente et non sa direction.
La ligne obtenant la valeur maximum lors de ce calcul est la ligne de plus forte courbure.

Ce calcul n'est pas très robuste et peut être perturbé par de nombreux facteurs. C'est cependant un moyen simple de déterminer un centre pour des images qui sont relativement rares (environ 3% des empreintes).

3. Calcul des paramètres locaux

Calcul de différents paramètres locaux (Ploc). Ces paramètres sont : orientation par bloc, qualité par bloc.

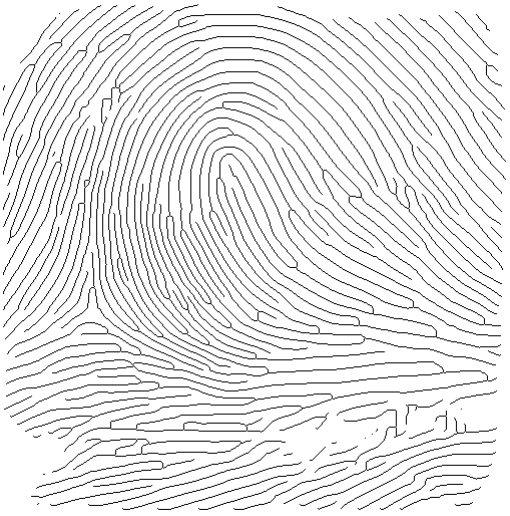


4.1 **Détermination des paramètres du tableau d'informations locales :** On calcule à partir du masque les dimensions des tableaux de paramètres locaux, leur centrage par rapport à l'images, et d'autre informations permettant de calculer simplement les paramètres en 4.2.

4.2 **Remplir le tableau d'informations locales :** deux tableaux sont remplis avec des informations concernant l'orientation et la qualité local de l'image sur des blocs de taille Tblocal (24 habituellement). Ces informations sont calculées en effectuant une moyenne des orientations (Ori2) et qualité (Qual2) des pixels de chaque bloc.

3. Extraction des minuties

L'image (Ia) est binarisée et squelettisée et les minuties de l'image sont extraites. Un certain nombre de post-traitements visant à supprimer les fausses minuties sont appliqués avant de stocker celles-ci dans le gabarit.



Is : Image squelettisée (taille:544x544)

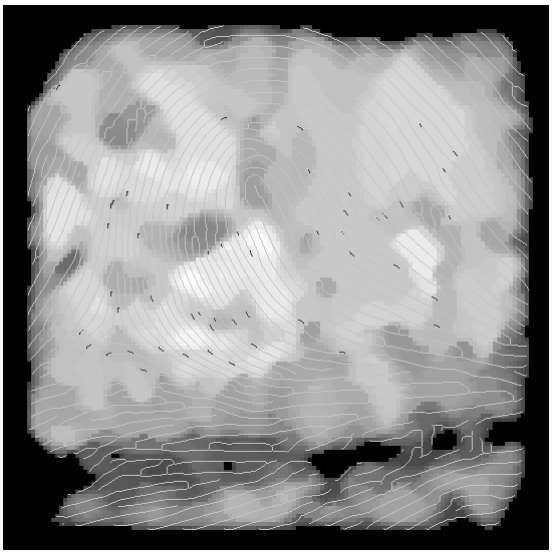


Image représentant la qualité de l'empreinte, le squelette des lignes d'empreinte ainsi que les minuties et leur orientation.

Squelettisation de l'image :

L'algorithme de squelettisation utilisée dans ce BioEngine est basé sur l'algorithme NWG, décrit notamment dans une publication nommée « A note on the Nagendraprasad-Wang-Gupta thinning algorithm » et publié dans la revue « Digital Signal Processing 3, page 97-102 ».

Peu de modifications ont été apportées à l'algorithme de départ si ce n'est 2 passes supplémentaires en fin de squelettisation afin de nettoyer correctement le squelette en vue de la détection des minuties (suppression de certains pixels empêchant la détection de minuties de type « bifurcation »).

Nous n'allons pas décrire en détail l'algorithme NWG puisqu'il est correctement documenté dans la publication citées ci-dessus. Notons juste que cet algorithme basé sur des masques pour décider quels pixels éliminer est particulièrement rapide.

Détection des minuties :

Durant cette phase, chacun des pixels du squelette est analysé. Le nombre de « voisins » et de « transitions » dans le voisinage 3x3 de chaque pixel est calculé.

Définitions :

Voisins : le squelette est une image globalement blanche (valeur=255) dont les points appartenant au squelette sont représentés en noir (valeur=0). Pour un point du squelette (dont la valeur est 0), on compte dans le voisinage 3x3 de celui-ci le nombre de points ayant également la valeur zéro. Ces points sont les voisins du point analysé.

Transistions : pour un pixel de squelette donné, le nombre de transitions est le nombre de changements pixel blanc -> pixel noir lorsque l'on parcourt la courbe numérique fermée formée par le voisinage 3x3 du pixel donnée.

Exemple:

On a représenté ci-dessous un point P et son voisinage 3x3.
Ce point P a 3 « voisins » (d0,d3 et d6) et possède 3 transitions (d2->d3, d5->d6 et d7->d0).

d2	d3	d4
d1	P	d5
d0	d7	d6

Détection des minuties à partir des « voisins » et « transitions » :

On détecte les minuties à partir de ces deux règles simples:

- Si un point du squelette possède 1 « voisin » (et donc obligatoirement 1 « transition »), alors c'est une minuties de type « fin de ligne ».
- Si un point du squelette possède 3 « voisins » et 3 « transitions », alors c'est une minutie de type « bifurcation ».

Le parcourt du squelette et l'application de ces 2 règles simples permet de détecter la position et le type de toutes les minuties de l'image.
Cependant à cause du bruit présent dans l'image d'origine (et malgré l'amélioration préalable de l'image), des « fausses » minuties sont détectées. Il faudra donc effectuer des post-traitements afin d'en éliminer le maximum.

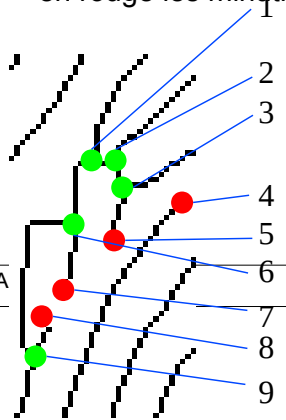
Pour faciliter les post-traitements, une étape supplémentaire est ajoutée lors de la détection des minuties : Pour chaque minuties détectée, on recherche les autres minuties présentes dans un voisinage de taille déterminée. L'analyse de ces minuties « voisines » permettra de supprimer les « fausses » minuties.

Détection des minuties voisines :

Comme expliqué plus haut, la détection des minuties voisines permet de préparer la tâche de nettoyage des fausses minuties.

Pour cela on effectue, pour chaque minutie détectée dans l'étape précédente (et quel que soit son type) une recherche des minuties « proches » de la minutie en cours d'analyse et connectées à celle-ci par une ligne du squelette.

Prenons pour illustration la figure ci-dessous représentant le squelette d'une partie (de mauvaise qualité) de notre image témoin. Sont représentés :
- en vert les minuties de type « bifurcation »
- en rouge les minuties de type « fin de ligne »



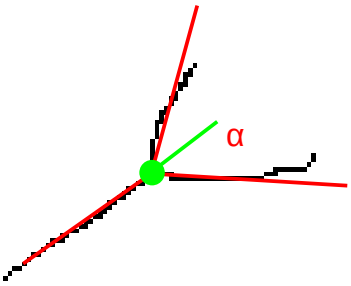
L'analyse de la minutie numéro 1 se fera de la façon suivante : On suit chacune des trois branches partant de la minutie sur une longueur déterminée (par exemple 16 pixels). Si sur cette longueur une minutie est détectée, elle est ajoutée à la liste de minuties voisines de la minutie en cours de traitement. Dès qu'une minutie est rencontrée sur une des branches, on arrête la recherche sur

cette branche et on passe à la suivante. Ainsi pour la minuties numéro 1, les minuties suivantes seront considérées comme voisines : 2 et 6 (mais pas la numéro 3 car bien que proche de la minuties n°1, elle est séparée de celle-ci par la minutie numéro 2).

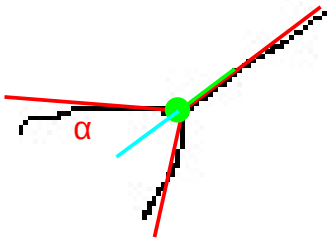
En appliquant cette méthode à chaque minutie détectée, on remplit une structure de données permettant un post-traitement plus aisé.

Détection de l'orientation des minuties:

Cette dernière étape de traitement consiste à utiliser conjointement les informations contenues dans la carte d'orientations (pour obtenir une orientation de minuties entre 0 et 180°) et la structure du squelette de l'empreinte (afin de compléter l'orientation pour la définir entre 0 et 360°).



Cas 1 : l'angle alpha et l'orientation de la minutie (en vert) sont cohérents. Il n'y a pas de modification à apporter.



Cas 2: l'ange alpha et l'orientation de la minutie (en vert) sont opposés. Il faut corriger l'orientation de la minutie (en bleu clair).

Nettoyage des « fausses » minuties :

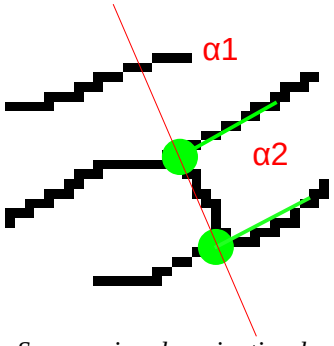
Comme indiqué plus haut, le « nettoyage » des minuties consiste à :

- Supprimer les minuties qui sont trop proches les unes des autres.
- Reconnecter le squelette aux endroit ou ses lignes sont proches et alignées et supprimer les minuties qui constituaient les extrémités des lignes avant re-connexion.

Suppression des minuties « parasites » :

La suppression des « fausses » minuties s'effectue selon les règles suivantes :

- Si la minutie (quel que soit son type) ne possède aucune minutie voisine, alors on la garde.
- Si la minutie possède des voisines et qu'elle est de type « bifurcation » : On calcule l'angle entre la ligne droite (imaginaire) reliant les deux minuties. On détermine ensuite les angles α1 et α2 que forme cette ligne et l'orientation de chaque minutie. Si cet angle est trop important, c'est que la ligne qui relie les deux minuties est une ligne parasite. On supprime alors les deux minuties.



Suppression des minuties de type bifurcation.

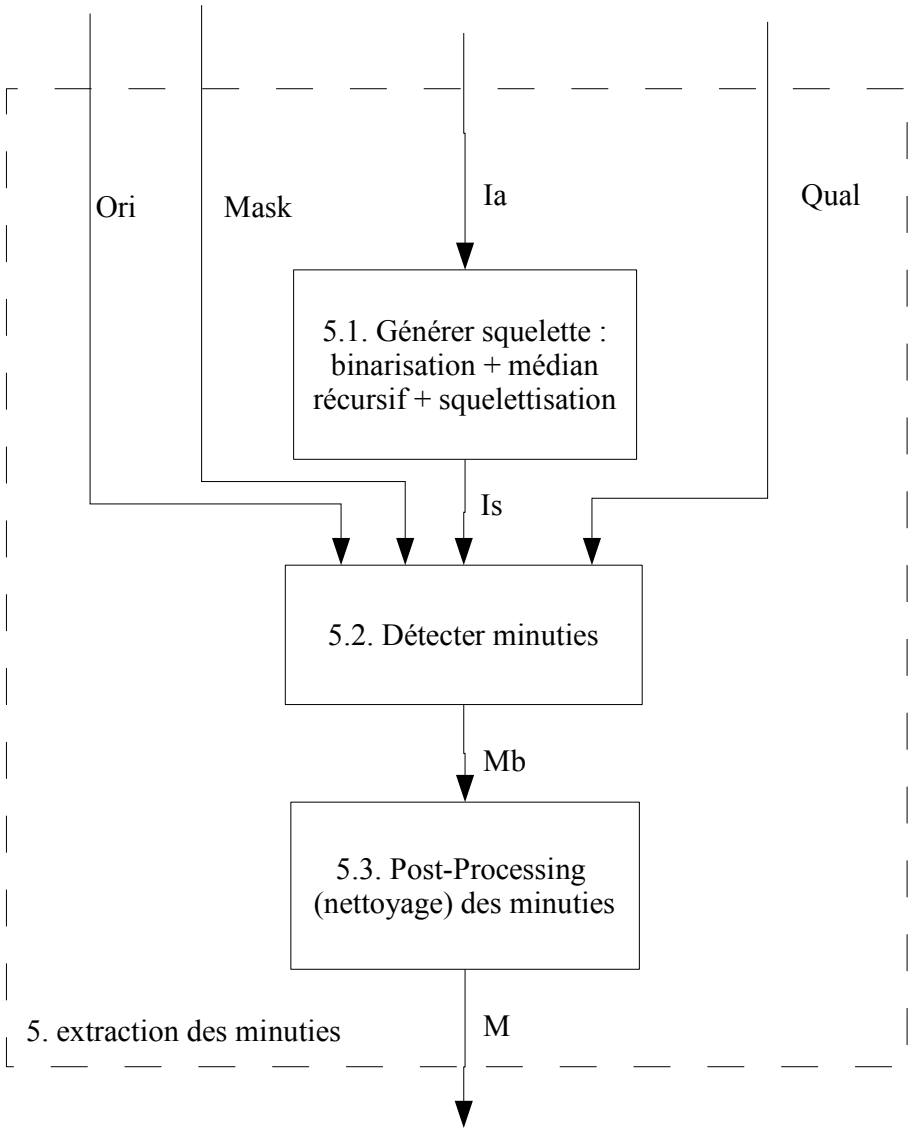
- Si la minutie possède des voisines et est de type « fin de ligne » : on la supprime. Le traitement des minuties étant effectué récursivement et en profondeur d'abord, une fin de ligne ne sera supprimée que si les bifurcations qui étaient ses voisines n'ont pas été supprimées).

Reconnexion du squelette :

La reconnexion des lignes du squelette s'effectue également à partir d'un jeux de règles simples :

- On ne modifie pas les minuties de type « bifuraction »

- Si on rencontre une minutie de type « fin de ligne », on recherche la ou les minuties située à une distance inférieure à un certain seuil (16 pixels par exemple). On calcule alors (comme lors de l'étape de suppression des « fausses » minuties) les angles α1 et α2 entre chacune des minuties analysées et la ligne virtuelle reliant celles-ci. Si cet angle est suffisamment petit, alors on considère que les minuties n'ont pas lieu d'exister car elle sont aux extrémités d'une ligne d'empreinte qui a été accidentellement « coupée » à cause du bruit présent dans l'image.



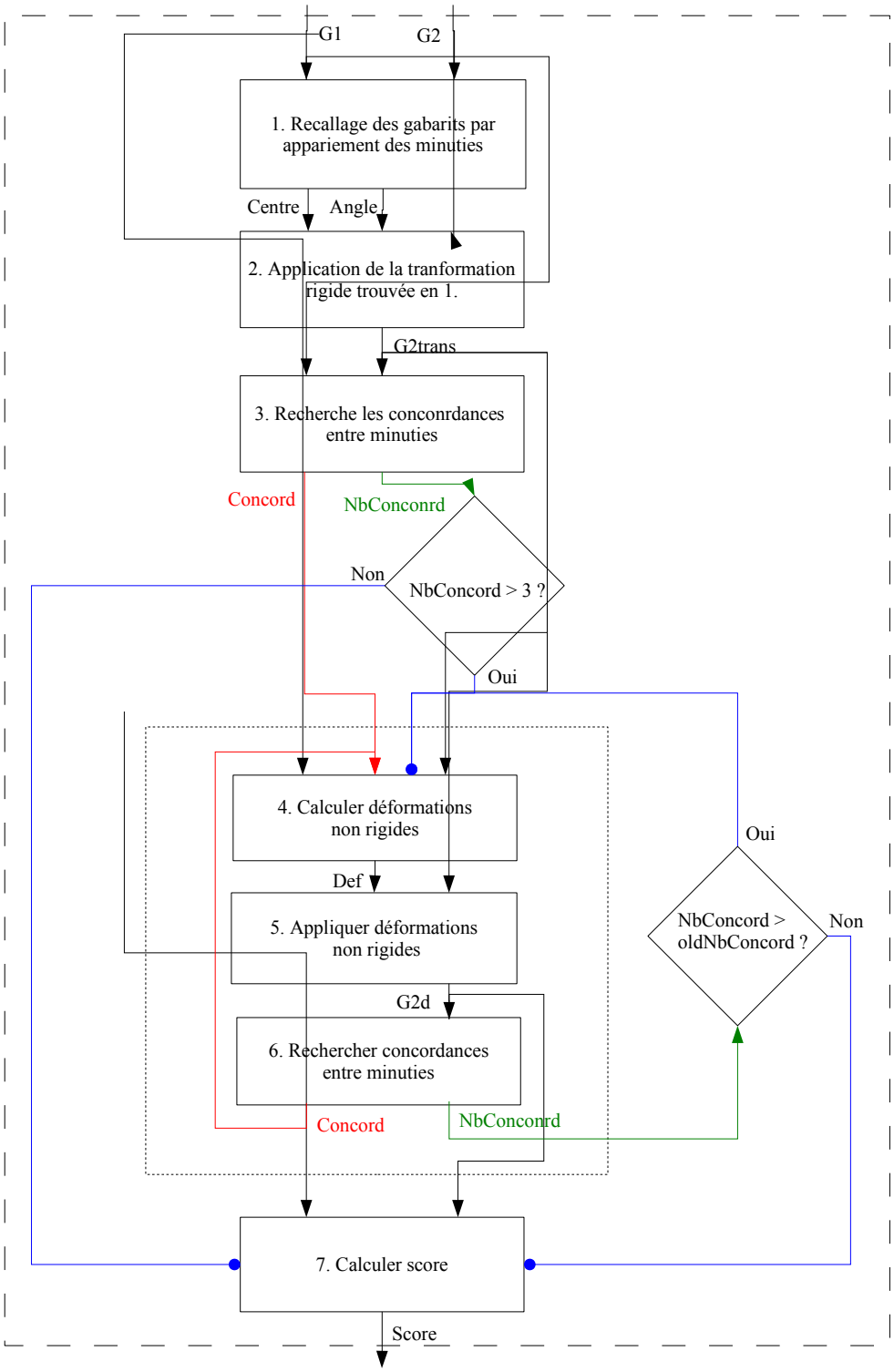
6. Stockage de toutes les informations dans le gabarit.

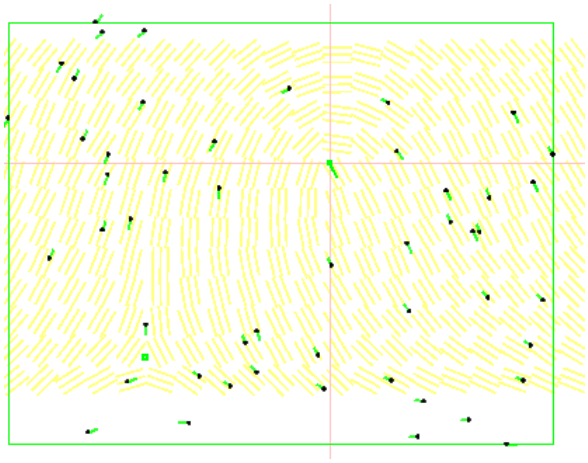
Cette suite de traitements permet d'obtenir un gabarit qui pourra ensuite être exploité par l'algorithme de matching.

Il est à noté qu'une autre fonctionnalité a été rajoutée. Celle-ci permet de construire un nouveau gabarit composite à partir de deux gabarits représentant la même empreinte. Cette fonction permet d'améliorer la qualité de l'enrôlement et éventuellement d'améliorer le gabarit au fur et à mesure des reconnaissances si celles-ci sont de bonne qualité. Elle est décrite au chapitre IV.

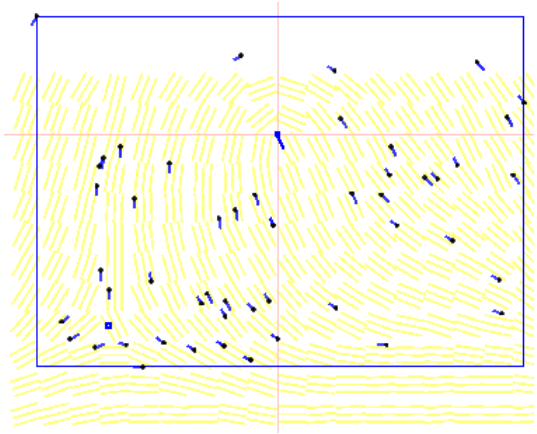
I. Architecture générale de l'algorithme de comparaison de gabarits (Authentification).

L'algorithme général est composé de 7 blocs. Trois de ces blocs sont exécutés à l'intérieur d'une boucle dont la condition d'arrêt permet de passer au calcul du score.


G1
G2
1. Recallage des gabarits par appariement des minuties
Centre
Angle
2. Application de la tranformation rigide trouvée en 1.
G2trans
3. Recherche les conconrdances entre minuties
Concord
NbConconrd
Non
NbConcord > 3 ?
Oui
4. Calculer déformations non rigides
Def
5. Appliquer déformations non rigides
G2d
6. Rechercher concordances entre minuties
Concord
NbConconrd
Oui
NbConcord > oldNbConcord ?
Non
7. Calculer score
Score



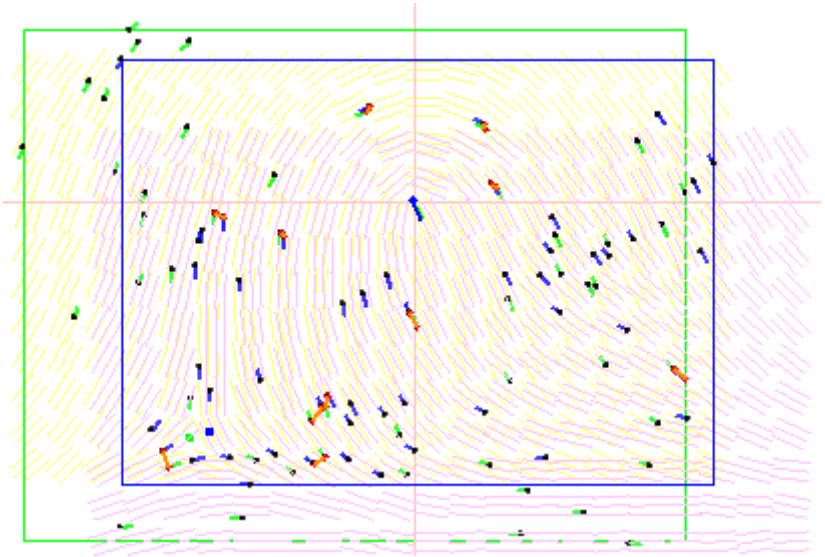
Représentation graphique du gabarit 1



Représentation graphique du gabarit 2 (provenant du même doigt que le gabarit 1)

Les fonctions des 7 blocs sont les suivantes :

1. **Recalage des gabarits :** On utilise la position et l'orientation de minuties pour recaler les deux gabarits. Pour aller plus vite, le recalage est effectué en deux fois : d'une manière assez grossière, puis plus précisément. On obtient en sortie un angle *(angle)* et les écarts dx et dy entre les deux centres *(centre)*.



Correspondances entre les gabarits 1 et 2 avant recalage : 11 minuties communes (en rouge). Les gabarits sont positionnés selon leur core principal (origine du repère).

Détails de la méthode :

<u>1ère passe de traitement :</u>

La structure principale de l'algorithme utilisé est la suivante :

Pour *Theta* de -*ThetaMax* a +*ThetaMax* par pas de *ThetaStep*
    Pour chaque minuties *M2* du gabarit *G2*
        Appliquer la rotation d'angle *Theta* a *M2*
        Pour chaque minuties *M1* du gabarit *G1*
            Si Abs(orientation(*M1*)-Orientation(*M2*)) < *SeuiOrientation*

Remplir les tableaux : *Dx*, *Dy*, *Val* et *Index*.
Fin Si
Fin Pour
Fin Pour
Remplir le tableau *Accu* à partir des information de *Dx*, *Dy*, *Val* et *Index*
Rechercher la meilleure translation pour la rotation *Theta* en cours à partir des donnée de *Dx*, *Dy*, *Val*, *Index* et *Accu* => on obtient une T*ranslation* plus une valeur de C*onfiance* en cette translation
Si *Confiance* > *ConfianceOk*
TranslationOk = *Translation*
ConfianceOk = *Confiance*
ThetaOk = *Theta*
Fin si
Fin Pour

Renvoyer *ThetaOk* et *TranslationOk*

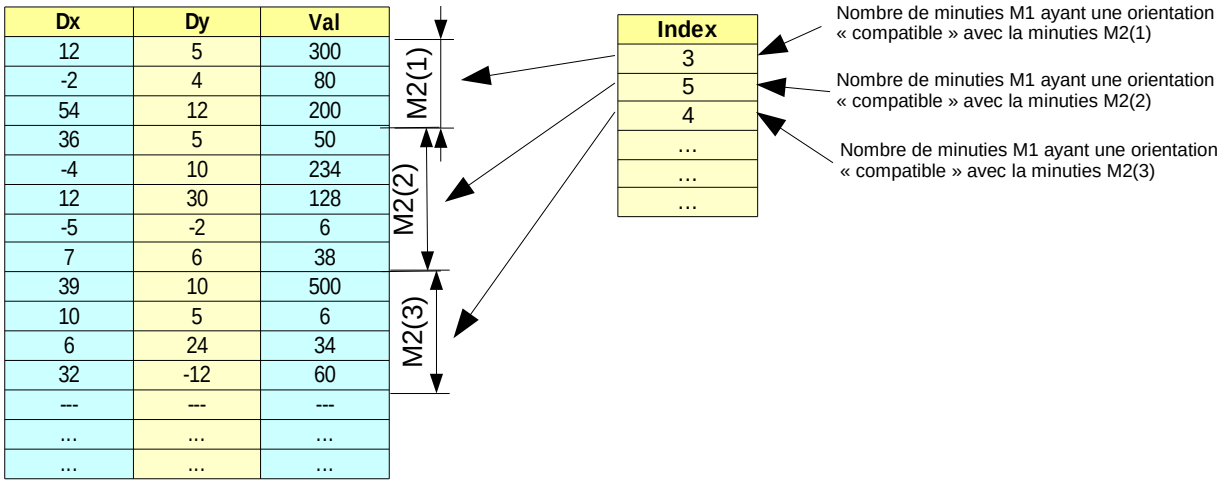
Descriptif des tableaux utilisés :

Dx et Dy : Tableau contenant les translations sur les axes des x et y entre une minutie M2 et une minutie M1 dont l'orientation est proche de celle de M2 (après application de la rotation théta à M2).

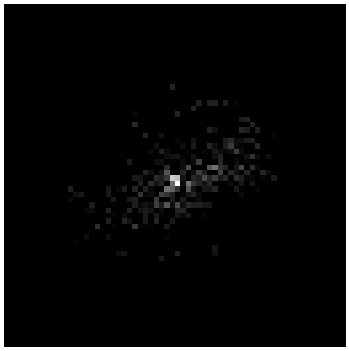
Val : Indice de confiance concernant Dx et Dy pour un couple M1(a),M2(b) donné. Est calculé en fonction de l'écart d'orientation entre M1 et M2 ainsi qu'a partir de la qualité de M1 et M2 (qualité calculée lors de la phase d'amélioration de l'image et stockée dans le gabarit avec chaque minutie).

Index : Tableau contenant, pour chaque minutie M2(a), le nombre de minuties M1 ayant une orientation proche la minutie M2(a).

Accu : Tableau à deux dimensions exprimant les « votes » pour différentes valeur de translation. Les valeurs de translation sont quantisées (avec un pas de 16 pixels), afin de permettre une détection plus robuste. Ce tableau est rempli à partir des informations de Dx, Dy et Val.



Exemple permettant de mieux comprendre les valeurs stockées dans les différents tableaux utilisés par l'algorithme de recalage..



Représentation graphique du tableau 2D Accu pour théta = 0 (taille: 64x64).

Détermination de la meilleure translation pour un angle Théta donné :

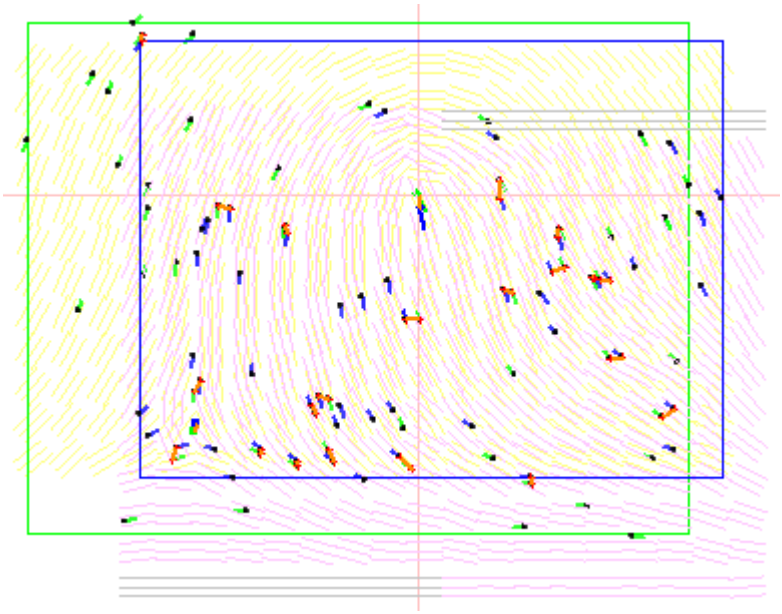
La détermination de la meilleure translation ne consiste pas en une simple recherche de maximum dans la tableau Accu.
En effet cette recherche ne serait pas vraiment exacte, car comme permet de le constater le tableau Index, on trouve lors de notre recherche de minutie M1 correspondant à M2(x), un certain nombre de minuties M1. Cela veut dire que les votes sont biaisés car pour chaque minutie M2, de nombreux votes sont effectués (alors qu'un seul n'est réellement correct).
Une méthode un peu plus évoluée est utilisée, afin de rendre la détection un peu plus robuste. Cette méthode consiste, pour chaque translation potentiellement acceptable, à parcourir la liste des minuties M1 correspondant potentiellement à chaque minutie M2 et a ne garder le vote que de la seule minutie valable.

La translation choisie (quel que soit théta) sera celle ayant obtenu le meilleur indice de confiance (calculé à partir de « vraie » paire M1,M2 uniques pour chaque translation).

<u>2ème passe de traitement :</u>

La deuxième passe de traitement effectue les mêmes opérations que lors de la première phase, mais cette fois-ci en prenant comme base la translation et la rotation trouvés lors de la 1ere phase. Le but des traitements est alors d'affiner ces valeurs afin d'obtenir une estimation de la rotation et de la translation plus précise.

2. **Application de la transformation :** On effectue la translation et la rotation calculés en 1. sur le gabarit G2. La rotation et la translation sont appliqués aux minuties, aux cores et deltas ainsi qu'aux informations de bloc locales (qualité et orientation). Les deux gabarits sont alors alignés. Le gabarit G2 ainsi transformé est appelé *(G2trans)*.



Correspondances entre les gabarits 1 et 2 après recalage : 21 minuties communes (en rouge). Corrections : +2 pixels en X, + 10 pixels en Y et +5° pour la rotation.

3. **Recherche des concordances entre minuties :** On recherche les minuties qui sont suffisamment proches et dont les orientations sont semblables.

Pour trouver les concordances, on parcourt, pour chaque minutie M2, l'ensemble des minuties M1 à la recherche d'un couple suffisamment semblable. On garde dans un tableau les couples M1,M2 précédents afin de ne pas associer la même minutie M1 à plusieurs minuties M2.

En sortie de cette fonction, on obtient une liste de *(nbConcord)* couples de minuties *(Concord)*. Chaque couple comprend une minutie du gabarit G1 et une minutie du gabarit G2trans.

Si le nombre de minuties qui concordent dans le bloc 3 est supérieur à 3 (moins de chance de minuties colinéaires), on effectue la boucle comprenant les blocs 4,5 et 6. Sinon, on calcule directement le score.

4. **Estimation des déformations non rigides :** A partir de la liste des minuties concordantes, on calcule, en utilisant la méthode des thin-plate splines un modèle de déformation non rigide (*Def*) qui pourra ensuite être appliqué à l'ensemble des minuties. Le but de cette étape est de pouvoir rapprocher des minuties qui normalement n'auraient pas matché tout en maintenant une certaine cohérence dans les déformations.

Le calcul effectué lors de cette étape de base sur le principe suivant :

A partir des minuties concordantes, on calcule un modèle approximant des thin-plate splines et permettant de « mapper » n'importe quel point du gabarit G2 vers le Gabarit G1.

La formule de ce « mapping » est la suivante :

$$f(x,y)=a_1+a_2x+a_3y+\sum_{i=1}^{n} w_i U(|P_i-(x,y)|)$$

avec $U(r) = r^2\log(r)$.

$a= [a_1, a_2, a_3]^T$ représente la partie affine de la transformation.

w représente la partie non linéaire de la déformations

P_i sont les points de référence utilisés pour générer le modèle TPS (c'est à dire les coordonnées des minuties concordantes).

Pour calculer les paramètres du modèle (c'est à dire les vecteurs a et w), nous utilisons une méthode permettant d'obtenir une approximation de celui-ci (avec un paramètre λ, permettant de régler la plus ou moins grande exactitude du modèle (λ = 0 correspond à une concordance exacte alors que des valeurs plus grandes permettent au modèle plus de liberté).

La valeur de λ utilisée dans l'algorithme est de 0.5. Ceci est un bon compromis pour simuler l'élasticité de la peau.

Calcul du modèle :

Trouver a et w consiste à résoudre le système suivant :

$$\begin{bmatrix} K+\lambda I & P \\ P^T & O \end{bmatrix} \begin{bmatrix} w \\ a \end{bmatrix} = \begin{bmatrix} v \\ o \end{bmatrix}$$

Avec :

$K_{ij} = U(|(x_i,y_i) – (x_j,y_j)|)$, sa taille est : n*n

la $i^{ème}$ ligne de P est : $(1, x_i, y_i)$ avec (x_i,y_i) les coordonnées du $i^{ème}$ point de contrôle

O est une matrice 3x3 remplie de zéros

o est un vecteur colonne de zéros

w est un vecteur colonne composé des paramètres w_i (que l'on recherche)

a est un vecteur colonne contenant a_1, a_2 et a_3

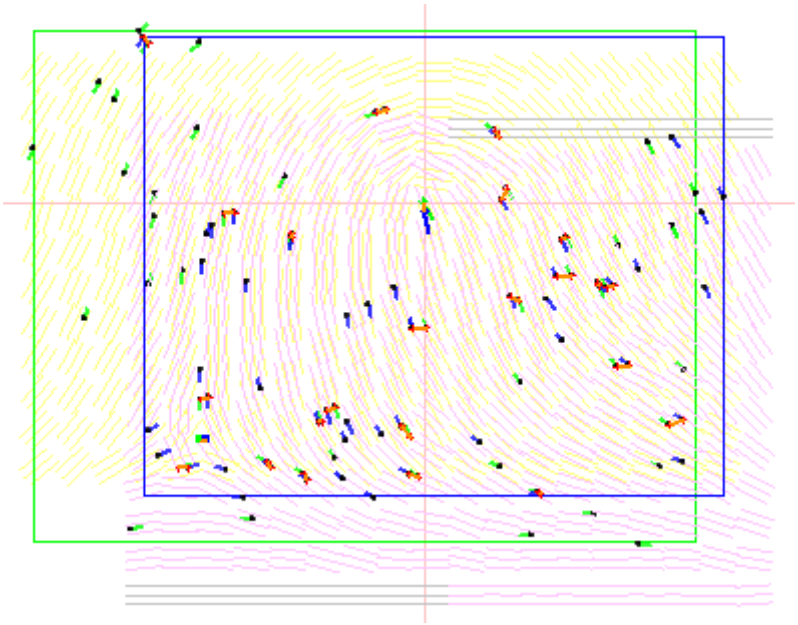
v est un vecteur colonne contenant soit les coordonnées en x des points de destination, soit les coordonnées en y des points de référence (la résolution se fait en deux étapes).

λ est le paramètre de régularisation, permettant de rendre la résolution du système plus ou moins exacte.

I est une matrice identité de taille n*n

Pour résoudre ce système, on utilise une décomposition LU.

5. **Application du modèle de déformation non rigide :** On applique le modèle calculé en 4 à toutes les minuties du gabarit G2trans. Le gabarit ainsi modifié est appelé *(G2d).*
   Après application de la déformation, on récupère une valeur *Diff* qui représente la quantité de déformation appliquée au gabarit G2. Si cette déformation est trop importante, c'est qu'il y a eu erreur lors du calcul de la déformation et qu'il faut annuler celle-ci. On récupère donc le gabarit G2 aligné et on passe à l'étape 7.



Correspondances entre les gabarits 1 et 2 après déformation non rigide : 24 minuties communes (en rouge).

6. **Recherche des concordances entre minuties :** on effectue à nouveau le même traitement qu'en 3 en espérant que le nombre de couple de minuties trouvé *(nbConcord)* soit supérieur à celui trouvé lors de l'étape 3 ou du tour de bouclé précédent. Si c'est le cas, on re-boucle pour affiner notre modèle de déformation.

Quand le nombre de couple de minuties trouvé ne change pas entre deux tours de boucle, on peut alors calculer le score

7. **Calcul du score :** Plusieurs éléments sont pris en compte lors du calcul du score. Le nombre de minuties qui concordent, leur distance relative, le fait qu'elles soient regroupées, le nombre de minuties qui ne matchent pas dans la zone commune aux deux gabarits, les orientations du motif des deux empreintes, leur qualité locale, etc. Tout ces éléments sont mélangés pour obtenir un *(score)*.

Détails du calcul du score :

Le calcul du score finale de comparaison de deux gabarits G1 et G2 est effectué à partir de 4 valeurs principales:

1. TxFreq : Un score de comparaison des fréquences moyennes des lignes d'empreinte de G1 et G2. Ce score est une valeur entre 0 et 1 (la valeur 1 signifiant que les deux empreintes ont la même fréquence moyenne).

2. matchScore : Score calculé à partir des minuties concordantes entre G1 et G2.

Ce score de concordance entre les gabarits G1 et G2, ayant en commun le set de minuties Mi (avec i de 0 à n) est calculé de la façon suivante :

$$matchScore(M_i) = \sum_{i=0}^{n} \frac{boostScore(i) * Q1_i * Q2_i}{1 + \frac{\Delta dist(i) + \Delta ori(i)}{distDivCoef}}$$

avec :

$M1_i$ et $M2_i$ les $i^{èmes}$ minuties concordantes de G1 et G2

$Q1_i$ et $Q2_i$ la qualité de $M1_i$ et $M2_i$ (dont les valeurs sont comprises dans l'intervalle [0...255])

Δdist(i) la distance entre $M1_i$ et $M2_i$ calculée suivant la formule suivante :

$$\Delta dist(i)=\sqrt{((x1_i-x2_i)^2+(y1_i-y2_i)^2)}$$

$(x1_i,y1_i)$ et $(x2_i,y2_i)$ étant respectivement les coordonnées de $M1_i$ et $M2_i$

Δori(i) la différence d'orientation entre $M1_i$ et $M2_i$ calculée de la manière suivante :

Δori(i) = | Ori1i – Ori2i | si | Ori1i – Ori2i | <= 180 °
= 360° - | Ori1i – Ori2i | sinon

distDivCoef est une constante de normalisation dont la valeur typique est 3

boostScore est une valeur calculée en fonction de la façon dont les minuties qui ne concordent pas sont disposées par rapport à celles qui concordent. Le rôle de boostScore est de favoriser les minuties (concordantes) située dans des zones ou pas ou peu de minuties non concordantes sont présentes, renforçant ainsi le score dans les zone de « bonne qualité » (en terme de concordance de minuties).

Calcul de boostScore :

Pour expliquer ce calcul, nous prendrons comme référence l'illustration ci-dessous.

Pour chaque couple de minuties concordantes $(M1_i,M2_i)$, on calcule un rectangle reliant le centre de ce couple de minutie au centre de chacun des autres couples de minuties concordantes $(M1_j,M2_j)$.

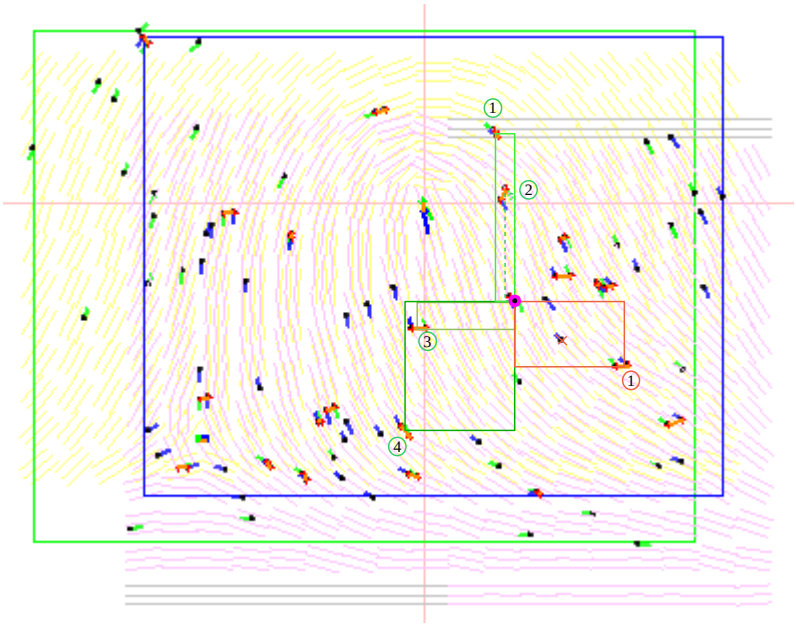
Dans notre exemple, le couple de minuties en cours d'analyse est le point rose. On a construit en vert les rectangles qui relient ce couple de minuties à d'autres couples de minuties, sans qu'aucune minutie non concordante n'appartienne aux rectangles formés.

On obtient 3 rectangles valides. (le rectangle reliant notre couple de minuties rose au couple n°3 n'est par pris en compte car il n'est pas assez large).

En rouge nous avons représenté un des rectangles ne pouvant pas être pris en compte car une minutie non concordante est située en son centre (une petite croix rouge est dessinée dessus).

Pour chacun des rectangles dont la construction est valable (pas de minutie non concordante à l'intérieur et rectangle de taille suffisante), on augmente boostScore de un.

Ainsi la valeur de boostScore pour notre couple de minuties rose serait de 1 (valeur de départ de boostScore) + 3 (nombre de rectangles « valides ») = 4.



Détails du calcul de boostScore

3. nonMatchScore : Score calculé en fonction des minuties non concordantes située dans la « zone commune » aux deux gabarits G1 et G2.

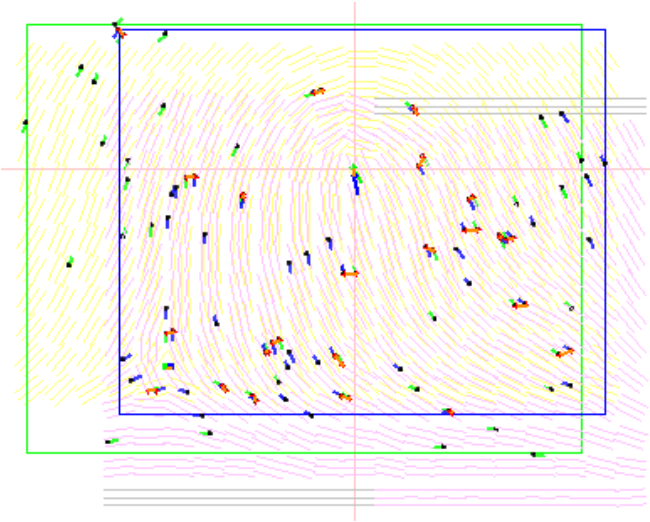
La formule de calcul de ce score est la suivante :

$$nonMatchScore = \sum_{i=0}^{n} \frac{QL2(M1_i) \times Q1_i^2}{15} + \sum_{i=0}^{n} \frac{QL1(M2_i) \times Q2_i^2}{15}$$

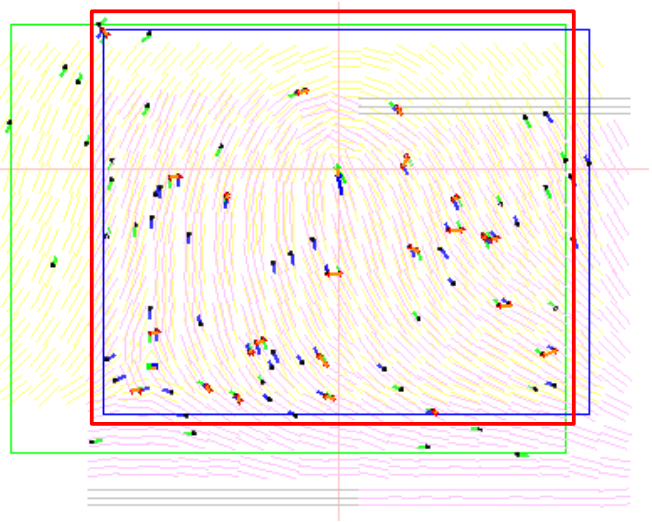
avec :

$Q1_i$ et $Q2_i$ la qualité de $M1_i$ et $M2_i$ (dont les valeurs sont comprises dans l'intervalle [0...255])

QL1(M) et QL2(M) la qualité local de bloc à la position de la ième minuties non concordante (dont les valeurs sont comprises dans l'intervalle [0...15]).



En vert la "boite" contenant toutes les minuties de G1. En bleu, la "boite" contenant toutes les minuties de G2



En rouge, la "boite" contenant toutes les minuties concordantes entre G1 et G2. Cette "zone commune" est l'intersection des "boites" de G1 et G2. Cette zone est légèrement agrandie pour prendre en compte les minuties (non concordantes) situées juste à la périphérie de la "zone commune" d'origine.

4. scoreOri : Score représentant la somme des différence d'orientation locale de bloc (en jaune et en rose sur les illustration représentant les gabarits G1 et G2) dans la « zone commune » entre les deux gabarits. Les valeurs de scoreOri sont comprises dans l'intervalle [0...4096].

Formule de calcul du score :

$$Score = scoreNormalize \times \frac{nbMatches}{maxNbMinutiae} \times \frac{maxScore \times txFreq \times matchScore}{(matchScore + nonMatchScore) * (1 + \frac{scoreOri}{oriMatchNormalize})}$$

$$si \quad Score > maxScore : Score = maxScore$$
$$si \quad Score < 0 : Score = 0$$

Avec :

scoreNormalize : paramètre de normalisation permettant d'avoir des seuils de sécurité cohérents (par exemple FAR=10-4 correspond à un seuil de 100), quels que soient les réglages du BioEngine.

nbMatches : nombre de minuties concordantes entre G1 et G2

maxNbMinutiae : nombre maximum de minuties supportées par le BioEngine (typiquement : 50)

maxScore : score maximal admissible (typiquement : 1000)

oriMathNormalize : paramètre de normalisation permettant d'ajuster l'influence de scorOri sur le score total (typiquement : 256)

III. Identification

Mécanisme de base :

Le mécanisme d'identification utilisé jusqu'à présent dans DpBioEngine n'est pas très complexe. Peu de temps de R&D a été affecté au développement d'une solution efficace d'identification, l'accent ayant plutôt été porté sur la partie authentification.

Le procédé mis en place est cependant efficace et répond à l'objectif suivant : améliorer le temps de réponse et réduire le parcourt de la base de donnée lorsque l'utilisateur est présent dans la base.
Lorsqu'il ne l'est pas, le système mis en place actuellement parcourt toute la base de donnée pour être sûr que le gabarit n'est vraiment pas dans la base (on peut néanmoins régler en partie ce problème en limitant la profondeur de recherche dans la base, mais au détriment des performances de reconnaissance.

Le principe de base utilisé est le suivant (des explications plus précises seront fournies plus loin dans ce document) :
Afin d'améliorer les temps de réponse en identification (lorsque l'empreinte candidate est censée être dans la base d'empreintes), on effectue un pré-tri des empreintes selon des critères globaux (fréquence moyenne des lignes de l'empreinte, présence, nombre et position des cores et deltas, motif du flot de lignes de l'empreinte autour du core principal, etc.).
Ce classement permet dans une majorité des cas de retrouver l'empreinte de référence correspondant à l'empreinte candidate en haut de la liste des empreintes triées, réduisant ainsi grandement le temps de recherche (car on arrête la recherche dès que l'on a trouvé une empreinte suffisamment ressemblante).

Afin d'améliorer encore les temps de réponse, on effectue une première recherche d'empreinte (après le tri) en utilisant un algorithme rapide d'alignement d'empreintes, basé sur la position des cores. Cet algorithme prend pour hypothèse que la position des cores principaux des deux empreintes est correcte et qu'il n'y a qu'à rechercher une rotation entre celles-ci. Ce principe permet d'accélérer grandement l'alignement mais ne fonctionne pas forcement si la position des cores n'est pas correcte (ce qui peut arriver en cas de mauvaise qualité d'image).

En cas d'échec de la première recherche, on effectue une recherche avec un algorithme d'alignement des empreintes plus lent, mais plus précis.

Détail des traitements :

L'algorithme (simplifié) utilisé pour effectuer une identification est le suivant :

```
Générer les données d'indexation coreDist et noCoreDist

Trier les gabaritsDeReference en fonction de coreDist

gabaritTrouvé = -1

Pour chaque gabarit Gr appartenant au tableau gabaritsDeReference trié
        si coreDist(Gr) > 300
                Sortir de la boucle
        fin si
```

```
        Calcul Rapide du score entre Gr et Gc

        si score > 2*seuilDeSécurité
                renvoyer Gr et score
        fin si

        si score > scoreMax
                gabaritTrouvé = Gr
                scoreMax = score
        fin si
Fin Pour

Si  gabaritTrouvé != -1
        Si Calcul  du score entre gabaritTrouvé et Gc > seuilDeSécurité
                renvoyer  gabaritTrouvé et score
        Fin si
Fin si

Trier les gabaritsDeReference en fonction de noCoreDist

Pour chaque gabarit Gr appartenant au tableau gabaritsDeReference trié
        Calcul  du score entre Gr et Gc

        si score > 2*seuilDeSécurité
                renvoyer Gr et score
        fin si

        si score > scoreMax
                gabaritTrouvé = Gr
                scoreMax = score
        fin si
Fin Pour

Si  gabaritTrouvé != -1
        Si Calcul  du score entre gabaritTrouvé et Gc > seuilDeSécurité
                renvoyer  gabaritTrouvé et score
        Fin si
Fin si
```

Légende :

- Calcul des information d'indexation
- Recherche rapide de gabarits en utilisant les informations des « cores » des empreintes
- Recherche plus lente, n'utilisant pas les informations de cores des empreintes

Nous allons maintenant décrire chaque variable utilisée dans cette algorithme simplifié et expliquer si nécessaire comment calculer celles-ci.

- ***GabaritsDeReférence* :** tableau contenant les gabarits avec lequel on veut comparer notre gabarit candidat. C'est la base de données des gabarits qui contient tous les

gabarits stockés lors de l'enrôlement des utilisateurs du système biométrique.

- ***Gr* :** un gabarit parmi les N gabarits stockés dans le tableau GabaritsDeReférence
- ***Gc* :** le gabarit candidat dont on veut savoir s'il correspond à un gabarit stocké dans le tableau GabaritsDeReférence.
- ***GabaritTrouvé* :** variable temporaire permettant de stocker le gabarit (ou son index) que l'on considère être le plus proche de Gc. A la fin de l'execution de l'algorithme, si le score de comparaison entre Gc et GabaritTrouvé est supérieur au seuilDeSécurité, alors on renverra GabaritTrouvé comme le gabarit correspondant à Gc.
- ***SeuilDeSécurité* :** Valeur permettant de déterminer au delà de quel score on considère que deux gabarits correspondent.
- ***Score* :** valeur indiquant la similarité entre deux gabarits. Plus celui-ci est élevé et plus les gabarits sont proches.
- ***ScoreMax* :** variable temporaire permettant de stocker la valeur de score maximal atteinte tout au long du processus d'identification.
- ***coreDist* :** Valeur d'indexation calculée à partir de données dépendant et ne dépendant pas de la position des cores de deux gabarits en cours d'analyse. CoreDist n'est calculé que si Gr et Gc possèdent au moins un core valide.

Pour chaque couple (Gr,Gc) coreDist est calculé de la manière suivante :

coreDist(Gr,Gc) = meanFreqScore(Gr,Gc) + scoreIndex(Gr,Gc) + 100 / (distCoreScore(Gr,Gc) + distDeltaScore(Gr,Gc))

avec :

- meanFreqScore(Gr,Gc) = 10 * | meanFreq(Gr)-meanFreq(Gc) | si meanFreq(Gr) != 0 et meanFreq(Gc) != 0
  = 1000 sinon

- meanFreq : la fréquence moyenne d'écartement des lignes de l'empreinte (calculée lors de l'amélioration de l'image et stockée dans le gabarit).
- scoreIndex : score calculé à partir de la différence des « signatures » calculées autour du core principale des empreinte candidate et de référence.

La signature est calculé de la façon suivante :

$$sign(i)=Orientation(X(i),Y(i))-Orientation(X(i-1),Y(i-1))$$

$$avec$$

$$X(i)=xCore+distCore*\cos(thetaCore+i*N/360)$$
$$Y(i)=yCore+distCore*\sin(thetaCore+i*N/360)$$

$$si\ \ sign(i)>180°:sign(i)=360-sign(i)$$
$$si\ \ sign(i)<-180°:sign(i)=360+sign(i)$$

Et : i : appartenant à l'intervalle [0...N]
N : le nombre d'éléments dans la signature
Orientation(x,y) : orientation des lignes de l'empreinte à la position (x,y)
(xCore,yCore) : position du core principal de l'empreinte.
ThetaCore : orientation du core principal de l'empreinte.

Cette signature est une valeur relativement indépendante de la position et l'orientation de l'empreinte (à condition que la position et l'orientation du core soit correctement détectée). Comparer deux signatures permet donc théoriquement d'effectuer une mesure de similarité partielle des deux empreintes.

- distCoreScore : Valeur de similarité des cores de Gr et Gc, calculée uniquement si Gr et Gc ont deux cores chacun. Dans ce cas, on mesure la distance dGr séparant les deux cores de Gr et la distance dGc séparant les deux cores de Gc. Si |dGr-dGc| < 12 alors distCoreScore = 1.

- distDeltaScore : Valeur de similarité des Deltas de Gr et Gc, calculée uniquement si Gr et Gc ont au moins un core et un delta chacun. Dans ce cas, on mesure la distance dGr séparant le core principal et le 1er delta de Gr et la distance dGc séparant le core principal et le 1er delta de Gc. Si |dGr-dGc| < 12 alors distDeltaScore = 1.

- ***NoCoreDist*** **:** Valeur d'indexation calculée à partir de données ne dépendant pas de la position des cores des deux gabarits en cours d'analyse.

Si nbCores(Gr) != 0 et nbCores(Gc) != 0

NoCoreDist(Gr,Gc) = coreDist(Gr,Gc)

Sinon

NoCoreDist(Gr,Gc) = meanFreqScore(Gr,Gc) + 150

IV. Amélioration des gabarits: génération de gabarit composite

Le principe de cette fonctionnalité est le suivant : on peut demander au BioEngine d'essayer, lors de chaque vérification biométrique, d'améliorer le gabarit de référence stockée dans la base de données à partir des informations contenues dans le gabarit de l'empreinte candidate.

Bien entendu, pour des raisons de sécurité et fiabilité cette opération n'est effectuée que si le score obtenu lors de la comparaison est élevé. L'amélioration n'est réalisée que si ce score est deux fois supérieur au seuil de sécurité définissant le niveau d'acceptation lors d'une

reconnaissance. Ainsi on améliore l'image que lorsque l'on est sûr du résultat.

Les améliorations effectuées sur le gabarit sont les suivantes :

- Ajustement de la fréquence moyenne des lignes de l'empreinte
- Validation de la présence des cores et deltas dans l'empreinte et affinage de leur position.
- Amélioration (ajouts et corrections) des données locales de l'empreinte (orientation et qualité par bloc).
- Suppression des fausse minuties et renforcement de la qualité des « vraies » minuties
- Ajout de minuties manquantes.