



André BERGUES-LAGARDE

1^{ère} année STI

Projet d'application

Implémentation du calcul de
descripteurs biométriques
dans un programme Java

Rapport de projet

Pilotes du projet :

Anas Abou El Kalam
Christophe Rosenberger

Année universitaire 2004 – 2005

Introduction

Au cours de ce compte rendu, le travail qui a été effectué sur le calcul de descripteurs biométriques à partir des moments de Zernike sera présenté. Seul les points importants seront abordés de façon à ne pas surcharger inutilement l'exposé.

Dans un premier temps, une partie sur l'implémentation du programme sera réalisée. Nous y parlerons de la façon dont a été interfacé du Java avec du C et des modifications qui ont été apporté au programme C fourni. Dans un deuxième temps, différents tests seront réalisés sur les 2 logiciels de manière à montrer leurs performances. Enfin, les différents problèmes et les perspectives de ce projet seront abordés.

SOMMAIRE

INTRODUCTION	1
I IMPLEMENTATION DU PROGRAMME	3
I.1) REALISATION DE LA LIAISON EN JAVA ET C	3
I.1.1. <i>Présentation</i>	3
I.1.2. <i>Pourquoi utiliser cette solution ?</i>	3
I.1.3. <i>Mise en œuvre</i>	3
I.1.4. <i>Implémentation</i>	4
I.2) RECODAGE DU CALCUL DES DESCRIPTEURS	4
I.2.1. <i>Fichier d'inclusions</i>	4
I.2.2. <i>Problème des variables globales</i>	Erreur ! Signet non défini.
I.2.3. <i>Utilisation des boucles</i>	Erreur ! Signet non défini.
I.2.4. <i>Coefficients biométriques</i>	Erreur ! Signet non défini.
II DETAILS SUR LES TESTS ET EXPERIENCES REALISES	ERREUR ! SIGNET NON DEFINI.
II.1) COMPARAISON DU PROGRAMME PIZ ET JAVA SUR UN ECHANTILLON	ERREUR ! SIGNET NON DEFINI.
II.1.1. <i>Expérience</i>	Erreur ! Signet non défini.
II.1.2. <i>Résultats</i>	Erreur ! Signet non défini.
II.1.3. <i>Fichiers utilisés</i>	Erreur ! Signet non défini.
II.2) MISE EN EVIDENCE D'UN BUG DE PIZ	ERREUR ! SIGNET NON DEFINI.
II.2.1. <i>Expérience</i>	Erreur ! Signet non défini.
II.2.2. <i>Résultats</i>	Erreur ! Signet non défini.
II.2.3. <i>Fichiers utilisés</i>	Erreur ! Signet non défini.
II.3) PROBLEME DE TAUX DE RECONNAISSANCE	ERREUR ! SIGNET NON DEFINI.
II.3.1. <i>Expérience</i>	Erreur ! Signet non défini.
II.3.2. <i>Résultats</i>	Erreur ! Signet non défini.
II.3.3. <i>Fichiers utilisés</i>	Erreur ! Signet non défini.
II.4) REALISATION DE TEST SUR DES VISAGES	ERREUR ! SIGNET NON DEFINI.
II.4.1. <i>Expérience</i>	Erreur ! Signet non défini.
II.4.2. <i>Résultats</i>	Erreur ! Signet non défini.
II.4.3. <i>Fichiers utilisés</i>	Erreur ! Signet non défini.
III PROBLEMES ET PERSPECTIVES	ERREUR ! SIGNET NON DEFINI.
III.1) PROBLEMES RENCONTRES	ERREUR ! SIGNET NON DEFINI.
III.1.1. <i>Taux de reconnaissance faibles</i>	Erreur ! Signet non défini.
III.1.2. <i>Perte d'information lors du passage en Java</i>	Erreur ! Signet non défini.
III.1.3. <i>Problème de bornes</i>	Erreur ! Signet non défini.
III.2) PERSPECTIVES	ERREUR ! SIGNET NON DEFINI.
CONCLUSION	ERREUR ! SIGNET NON DEFINI.
ANNEXES	ERREUR ! SIGNET NON DEFINI.
ARCHIVES	ERREUR ! SIGNET NON DEFINI.
GUIDES	ERREUR ! SIGNET NON DEFINI.

I Implémentation du programme

I.1) Réalisation de la liaison en Java et C

I.1.1. Présentation

Afin d'interfacer le code Java de notre application avec un programme de calcul des descripteurs en C, nous allons utiliser la librairie JNI. La JNI, littéralement Java Native Interface technologie, permet d'implémenter du code natif dans une classe Java.

I.1.2. Pourquoi utiliser cette solution ?

Plusieurs possibilités s'offraient pour mettre en place le calcul des descripteurs biométriques (recodage complet du code en Java, interfaçage avec Matlab ...).

Finalement le choix de la solution JNI s'est effectué pour :

- faciliter la programmation,
- utiliser des composants éprouvés déjà existants,
- permettre un développement indépendant entre la partie biométrique et la partie Javacard (les chercheurs pourront continuer à modifier le code C et recompiler la DLL sans que cela influe sur notre projet),
- pour des raisons de performance.

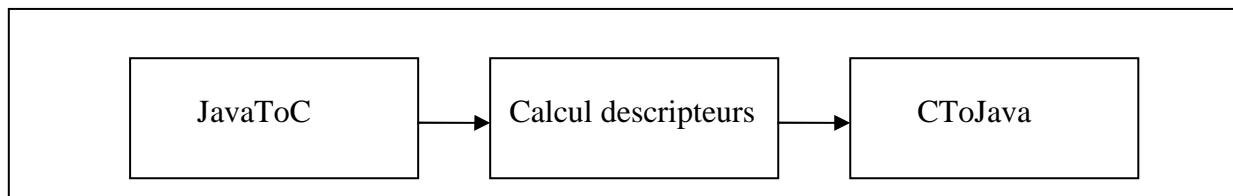
Cependant, l'utilisation de cette technologie apporte un inconvénient majeur, elle annule la portabilité du code Java. En effet, le calcul des descripteurs se fera dans une librairie DLL, hors les fichiers DLL n'existe que sous Windows.

I.1.3. Mise en œuvre

Tout d'abord, l'image va être chargée en java, puis transformée en niveau de gris si cela est nécessaire. Ensuite, une classe Java fera appel à une fonction native utilisant la DLL.

Cette fonction aura pour but de

- transformer l'image « java » en image « C ». Dans la pratique, cela consistera à convertir un tableau de 'char' à double dimension en son homologue en C,
- réaliser le calcul des descripteurs à partir du tableau précédent,
- renvoyer les descripteurs au processus Java pour qu'il puisse continuer son traitement. Les descripteurs seront stockés dans un tableau à simple dimension dans des variables à virgules flottantes.



Les 3 étapes évoquées précédemment constitueront les phases majeures de notre code, ainsi nous aurons 3 grandes fonctions :

```
imageXY JavatoC(JNIEnv * env, jobjectArray image);  
float * ZernikeGlobal(NVG ** imageNG, TAILLE X, TAILLE Y);  
jdoubleArray CtoJava(JNIEnv * env, float * DescripteurC);
```

I.1.4. Implémentation

L'utilisation de JNI comporte plusieurs étapes :

- la déclaration et l'utilisation de la ou des méthodes natives dans la classe Java,
- la compilation de la classe Java,
- la génération du fichier d'en-tête avec l'outil javah,
- l'écriture du code natif en utilisant entre autre les fichiers d'en-tête fournis par le JDK et celui généré précédemment,
- la compilation du code natif sous la forme d'une bibliothèque DLL.

Toutes ces étapes seront réalisées à l'aide d'un fichier « compil.bat » afin de faciliter la programmation. La réalisation pratique du programme ne sera pas détaillée. Cependant, le code contenu dans le fichier « calculdescripteur.cpp » est entièrement commenté, de plus les fichiers html contenu dans le répertoire 'doc' détaillent de façon très claire l'utilisation de cette librairie.

I.2) Recodage du calcul des descripteurs

Dans cette partie, quelques petites remarques seront faites sur la partie d'amélioration du code de calcul des descripteurs. En effet, un code propre et commenté facilite considérablement la programmation et la mise à jour du logiciel. De plus, un programme mal codé sera beaucoup plus difficile à debugger qu'un programme propre.

I.2.1. Fichier d'inclusions

Au niveau des fichiers d'inclusions, quatre fichiers étaient vraiment importants pour le programme. Dans le programme PIZ, il y avait 23 fichiers inclus, même si des fichiers sont nécessaire pour gérer une interface graphique et un listage des répertoires, beaucoup de fichiers étaient inutiles.

Voici le détail des fichiers qui ont finalement été inclus dans l'implémentation du programme Java.

```
// spécifique à java  
#include <jni.h> // librairie JNI permettant d'interfacer du Code C et Java  
#include "DescripteursBiometriques.h" // définition de la fonction native  
  
// spécifique au programme C  
#include <stdio.h> // bibliothèque standard  
#include <complex> // permet de gérer des variables de type complexe (a + jb)  
#include <iostream>  
#include <math.h> // fonction pow
```

POUR DES RAISONS DE CONFIDENTIALITE, LA SUITE DU RAPPORT N A PAS ETE DIFFUSEE.