

IMT ATLANTIQUE

INSTITUT MINES-TELECOM

---

**Conception et implémentation  
en C++ d'une bibliothèque  
d'algorithmes d'extraction de  
descripteurs sur des images à  
des fins de recherches orientées  
contenu.**

---

*Auteur :*

Khalid MAJDOUB

*Encadrant :*

Dr. Didier GUERIoT

19 juin 2017

## Résumé

Le but de ce stage est de mettre en place une bibliothèque en langage C++, d'extracteurs de descripteurs opérant sur des images. Ces différents descripteurs permettent de caractériser certaines propriétés spécifiques de ces images et ainsi de permettre des traitements ultérieurs de recherche d'images basées sur leur contenu. Un ensemble de routines réalisant ces extractions existe déjà en Matlab et il était demandé de les ré-implémenter en C++.

Le stage s'est ainsi déroulé en 3 étapes. Tout d'abord, une recherche bibliographique sur les algorithmes existants a été nécessaire afin de faciliter leur compréhension et les implémentations actuelles. A partir des algorithmes ainsi extraits, la deuxième étape a consisté à les coder en C++ pour les rassembler sous la forme d'une bibliothèque logicielle architecturée de manière modulaire et évolutive. Finalement, un protocole de test a été mis en place afin de valider et de qualifier la nouvelle implémentation par rapport à celle existante, en travaillant sur plusieurs bases d'images et en comparant les résultats obtenus par rapport aux résultats antérieurs.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Contexte du stage . . . . .	5
1.2	Objectifs du stage . . . . .	6
1.3	Plan du rapport . . . . .	6
<b>2</b>	<b>Descripteurs basés sur les histogrammes</b>	<b>8</b>
2.1	Extracteur BlockHistogramme (BHDsc) . . . . .	8
2.1.1	Algorithme de l'extracteur . . . . .	9
2.1.2	Utilisation de la bibliothèque . . . . .	10
2.1.3	Mesure de similarité . . . . .	11
2.1.4	Validation de l'implémentation . . . . .	12
2.2	Extracteur Angular radial partitioning (ARP) . . . . .	13
2.2.1	Algorithme de l'extracteur . . . . .	13
2.2.2	Utilisation de la bibliothèque . . . . .	15
2.2.3	Mesure de similarité . . . . .	16
2.2.4	Validation de l'implémentation . . . . .	16
<b>3</b>	<b>Descripteurs basés sur les moments</b>	<b>17</b>
3.1	Descripteur Moments de Hu (HuMomDescriptor) . . . . .	18
3.1.1	Algorithme de l'extracteur . . . . .	18
3.1.2	Utilisation de la bibliothèque . . . . .	19
3.1.3	Mesure de similarité . . . . .	20
3.1.4	Validation de l'implémentation . . . . .	20
3.2	Extracteur Angular Radial Transformation (ART) . . . . .	23
3.2.1	Algorithme de l'extracteur . . . . .	24
3.2.2	Utilisation de la bibliothèque . . . . .	25
3.2.3	Mesure de similarité . . . . .	26
3.2.4	Validation de l'implémentation . . . . .	26
3.3	Extracteur Moments de Zernike . . . . .	27
3.3.1	Principe de calcul des moments de Zernike . . . . .	27
3.3.2	Algorithme de l'extracteur . . . . .	28
3.3.3	Utilisation de la bibliothèque . . . . .	30
3.3.4	Mesure de similarité . . . . .	30
3.3.5	Validation de l'implémentation . . . . .	31

<b>4</b>	<b>Architecture de la bibliothèque</b>	<b>32</b>
4.1	Description des composants de la bibliothèque . . . . .	33
4.2	Réflexions sur l'architecture logicielle proposée . . . . .	38
<b>5</b>	<b>Conclusion et perspective</b>	<b>39</b>

## Table des figures

1	Contexte & périmètre du stage. . . . .	7
2	Découpage d'une image en $N_b \times N_b$ régions ou blocs (ici $N_b = 3$ ). . . . .	9
3	Construction du vecteur des caractéristiques $V_c$ à partir des valeurs des histogrammes calculées sur chaque bloc de l'image (avec ici, $3 \times 3$ blocs et 10 bins par histogramme). . . . .	10
4	Définition des sections radiales-angulaires du cercle unité. . . . .	14
5	Image binaire de test représentant une chauve-souris . . . . .	21
6	image de test . . . . .	31
7	Diagramme de classe pour la hiérarchie des classes de descripteurs. . . . .	33
8	Diagramme de classe pour la classe <b>Similarity</b> . . . . .	35
9	Diagramme de classe pour la classe <b>Tools</b> . . . . .	35
10	Diagramme de classe pour la classe <b>DscDataBase</b> . . . . .	37

## Liste des tableaux

1	Statistiques des similarités obtenues entre les vecteurs des caractéristiques obtenus via Matlab et via C++, pour le descripteur <b>BlockHistogram</b> sur une base de données de 15 images . . . . .	13
2	Statistiques des similarités obtenues entre les vecteurs des caractéristiques obtenus via Matlab et via C++, pour le descripteur <b>ARP</b> sur une base de données de 15 images . . . . .	17
3	Moments de Hu calculés pour l'image de test en figure 5. . . . .	21
4	Moments de Hu pour une forme ovale sous différentes rotations . . . . .	22
5	Distances de Mahalanobis entre une base de chauves-souris et d'autres images de chauves-souris . . . . .	22
6	Distances de Mahalanobis entre une base de chauves-souris et des images de pommes . . . . .	23

7	Statistiques des similarités obtenues entre les vecteurs des caractéristiques obtenus via Matlab et via C++, pour le descripteur <b>ART</b> sur une base de données de 15 images . . . . .	27
8	Tableau illustrant la différence entre les quatre premiers moments de Zernike obtenu par notre code et ceux obtenu par le code matlab sur une image de test . . . . .	31
9	Premiers moments de zernike d'ordre 1 à 6 générés par notre implémentation pour une image sous différentes rotations . . .	32
10	Exemple de reconstruction d'une image de Lena à partir des moments de Zernike calculés via l'implémentation en C++. . .	32

# 1 Introduction

## 1.1 Contexte du stage

Grâce aux avancées technologiques de ces dernières années, en particulier dans le domaine du multimédia et de l'informatique, l'information numérique est au cœur de plusieurs secteurs d'activités dans le monde industriel, médical, scientifique... Ces avancées ont permis l'échange massif de données multimédias entre autres, avec le grand public. Ces volumes de données accumulés n'auront aucun intérêt s'il n'est pas possible de retrouver les informations pertinentes concernant un intérêt particulier, dans un temps raisonnable. Ce besoin suscite donc le développement de techniques de recherche d'informations spécifiques parmi un ensemble volumineux de données, et en particulier en termes de recherche d'images ciblées.

Parmi les approches les plus anciennes pour répondre à ce besoin, on retrouve la technique de recherche d'images à base de texte connue sous le nom TBIR<sup>1</sup> et apparue à la fin des années 70. Il s'agit d'indexer manuellement chaque image par des mots-clés décrivant leur contenu, puis de stocker ces images étiquetées ou bien juste les étiquettes, dans un système de gestion de base de données, qui permette l'interrogation et la recherche d'images à partir de combinaison de mots-clés.

Malgré le succès de cette approche, l'indexation manuelle par des mots-clés souffre de deux désavantages majeurs. Le premier est qu'elle demande beaucoup de main d'œuvre pour les grandes bases de données. Le deuxième est son manque de précision (et de reproductibilité) du à la subjectivité de la perception humaine [1][2]. De plus, une telle indexation ne peut jamais décrire le contenu d'une image, de manière exhaustive.

L'accroissement exponentiel des bases d'images oblige à dépasser les annotations manuelles. Afin de répondre à cette problématique, une autre approche baptisée RIC<sup>2</sup> ou CBIR<sup>3</sup> en anglais, a été introduite dans le début des années 80. Les approches de type CBIR visent à indexer les images en exploitant certaines caractéristiques reposant sur leurs contenus visuels, comme la couleur, la texture ou bien les formes présentes [3][4]. Aujourd'hui, suite à de nombreux travaux de recherche en CBIR, on peut trouver dans la littérature plusieurs algorithmes qui permettent d'extraire automatiquement

---

1. Text Based Image Retrieval  
2. Recherche d'Image par le Contenu  
3. Content Based Image Retrieval

d'une image, un vecteur de caractéristiques qui décrivent son contenu et qui permettent de ce fait, d'alimenter des systèmes de recherche d'images [5][6][7].

## 1.2 Objectifs du stage

Dans le cadre de ce stage, il est demandé d'implémenter en C++ une liste pré-établie d'extracteurs de descripteurs utilisés classiquement en CBIR et déjà codés en Matlab. Chaque extracteur permet donc la production d'un vecteur de caractéristiques (vecteurs de valeurs numériques) à partir d'une image. Ces extracteurs peuvent être vus comme des réducteurs de dimensions puisque la taille de l'image est généralement supérieure à celle d'un vecteur de caractéristiques. Les extracteurs étudiés dans ce stage peuvent être catégorisés en deux familles selon qu'ils se basent directement sur les histogrammes issus des images ou bien sur leurs moments. L'objectif principal de ces algorithmes est d'extraire des vecteurs de caractéristiques robustes à certaines transformations affectant les images comme des translations, des changements d'échelle ou des rotations.

Il est aussi demandé de choisir et d'implémenter pour chacun de ces descripteurs, une mesure de similarité qui indique un degré de ressemblance entre vecteurs numériques d'un même descripteur. Grâce à cette mesure de similarité, il sera possible de valider la qualité des algorithmes codés en C++ par comparaison des vecteurs numériques produits avec ceux obtenus par les implémentations Matlab. Par ailleurs, ces mesures de similarité peuvent servir directement d'outils pour des phases ultérieures de classification, d'indexation ou de recherche d'images à partir d'une base de données (images). La figure 1 résume le *workflow* correspondant au périmètre de ce stage.

## 1.3 Plan du rapport

Le corps de ce rapport va donc s'articuler en deux grandes parties. La première partie sera composée de deux sections (une par famille de descripteurs) qui vont présenter pour chaque descripteur :

- son algorithme de principe issu de recherches bibliographiques et d'études du code Matlab existant,
- son implémentation en C++ avec des exemples d'utilisation au travers de la bibliothèque proposée,
- une mesure de similarité adaptée et son implémentation en C++,

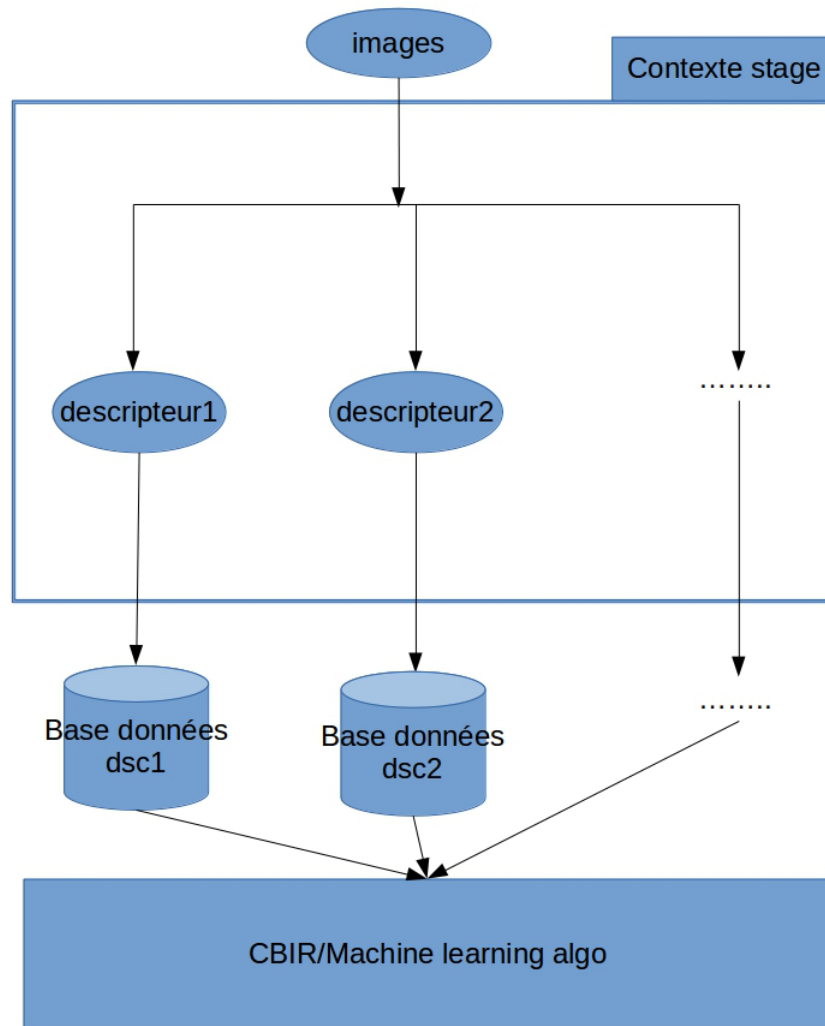


FIGURE 1 – Contexte & périmètre du stage.

- une validation de l’implémentation en C++ en se basant sur l’utilisation de la mesure de similarité entre les résultats d’extraction existants issus de Matlab et ceux obtenus avec la bibliothèque proposée.

La seconde partie détaillera l’architecture logicielle de la bibliothèque proposée en justifiant les choix d’implémentation en C++ en termes de modularité et d’évolutivité. Par ailleurs, une section spécifique sera consacrée à l’installation de cette bibliothèque.

Enfin, une conclusion dressera un bilan des résultats obtenus ainsi que



les perspectives envisagées que celles-ci portent sur la bibliothèque en elle-même, ou bien soient relatives à l'exploitation de cette bibliothèque dans des applications de classification ou de CBIR, par exemple.

## 2 Descripteurs basés sur les histogrammes

Les descripteurs basés sur histogrammes entrent dans la famille des descripteurs visuels, c'est-à-dire qu'ils exploitent les statistiques d'occurrence d'intensités des pixels de l'image pour en extraire des informations. Les valeurs des pixels d'une image ou d'une région d'images ne peuvent pas être exploitées directement. C'est pourquoi on extrait à l'aide d'algorithmes plus ou moins complexes des descripteurs visuels afin d'obtenir une représentation plus facile à utiliser. Pour pouvoir comparer des descripteurs visuels, il faut prendre en compte le fait que les régions visuelles considérées n'ont pas forcément la même taille. Il faut donc que la représentation des descripteurs visuels obtenue soit :

- discriminante : elle doit permettre de différencier des images différentes,
- invariante : deux régions de tailles différentes ou prises avec des luminosités différentes doivent avoir des représentations très proches,
- compacte : les espaces de grande dimension (une image par exemple) sont sensibles à *la malédiction de la dimension* et posent des problèmes de stockage des données. Il est donc intéressant de réduire au maximum la dimension des signatures.

Deux descripteurs de cette famille seront étudiés dans le cadre de ce stage :

- BHDsc : *Block Histogramme Descriptor*,
- ARP : *Angular Radial Partitioning*.

### 2.1 Extracteur BlockHistogramme (BHDsc)

Une représentation statistique d'une image est l'histogramme. Cet outil est plutôt insensible aux changements d'orientation, de taille et de position des régions, mais il ne capture pas les relations spatiales entre les pixels, et par conséquent a un pouvoir discriminant limité sur de telles propriétés.

Un histogramme peut être vu comme le nombre d'apparitions d'un élément dans un ensemble. Ainsi pour la construction d'un histogramme de descripteurs de couleurs, on passe par deux phases : d'abord, les couleurs sont quantifiées, puis les couleurs quantifiées sont dénombrées. Chaque composante du

vecteur de caractéristiques (appelée en anglais bin) donne la quantité d’une couleur présente dans l’image. Afin de rendre les histogrammes invariants à la taille de l’image ou des régions, on peut normaliser l’histogramme par le nombre total de pixels. Pour prendre en compte l’information spatiale dans l’histogramme, on peut ajouter la position des différentes régions de l’image, dans le cas où le calcul de l’histogramme ne s’applique pas sur l’ensemble de l’image mais sur des sous-régions [8].

### 2.1.1 Algorithme de l’extracteur

Dans cette approche, on commence par effectuer une division de l’image en  $B = N_b \times N_b$  blocs. Pour chaque bloc  $i$ , un histogramme  $hist_i$  de  $M_b$  bins est calculé. On suppose que les pixels de l’image varient entre 0 et 255 pour des images en niveaux de gris. Les valeurs de tous les bins de tous les blocs sont ensuite concaténées pour former le vecteur  $V_c$  des caractéristiques. Le descripteur  $V_c = (hist_1, hist_2, \dots, hist_B)$  produit par cet extracteur est donc de taille  $N_b \times N_b \times M_b$ .

hist1	hist2	hist3
hist4	hist5	hist6
hist7	hist8	hist9

FIGURE 2 – Découpage d’une image en  $N_b \times N_b$  régions ou blocs (ici  $N_b = 3$ ).

La figure 2 donne un exemple de découpage de l’image avec  $N_b = 3$  et  $M_b = 10$ . Ainsi, la matrice de l’image va être divisée en 9 blocs. Un histogramme de 10 bins est donc calculé pour chaque bloc. La figure 3 illustre comment est construit le vecteur des caractéristiques final à partir des 9 histogrammes.

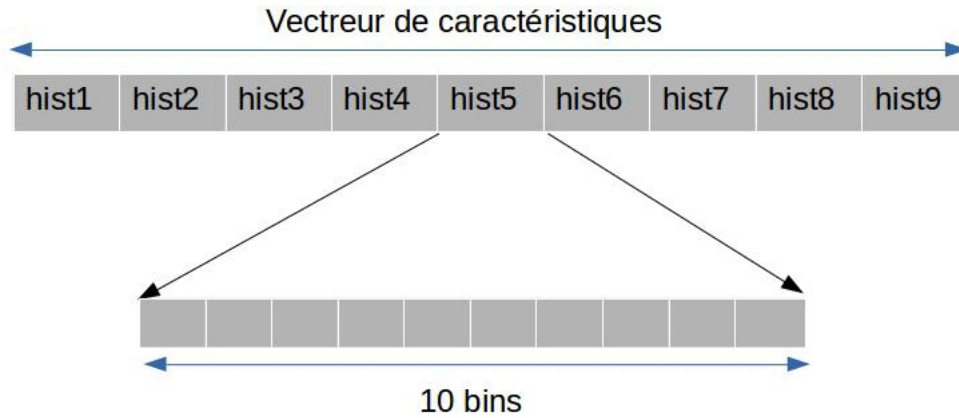


FIGURE 3 – Construction du vecteur des caractéristiques  $V_c$  à partir des valeurs des histogrammes calculées sur chaque bloc de l'image (avec ici,  $3 \times 3$  blocs et 10 bins par histogramme).

### 2.1.2 Utilisation de la bibliothèque

La bibliothèque **Extractor** livrée comporte une classe nommée **Block-HistDescriptor**. Le code suivant illustre comment lancer l'extraction du descripteur sur une image

```

1 #include "Extractor.h"
2
3 int nbrBlock = 3;
4 int nbrBin = 10;
5 BlockHistDescriptor myBHD(nbrBlock , nbrBin);
6 Mat inputImage;
7 inputImage = imread(argv[1],0);
8 vector<int> Vc;
9 myBHD.calcFeatures(inputImage , Vc);

```

Ainsi on commence par initialiser un objet **myBHD** de cette sous-classe en lui envoyant deux paramètres qui sont le nombre de blocs et le nombre de bins (ligne 5).

Pour calculer le vecteur des caractéristiques **Vc**, sur une image donnée **inputImage**, il suffit d'appeler la méthode **calcFeatures** du descripteur **myBHD** et de stocker le vecteur résultat dans un vecteur **Vc** juste créé (ligne 9). Le vecteur **Vc** doit être instancié comme un vecteur d'entiers (ligne 8).

Dans cet exemple, on charge l'image à analyser, dans une matrice **inputImage** en utilisant le routine **imread** de la bibliothèque **OpenCV** (ligne 7). Il faut noter que le 0 passé comme deuxième paramètre à **imread** signifie que l'image va être chargée en niveaux de gris (la plage des valeurs des pixels est de 0 à 255).

### 2.1.3 Mesure de similarité

**Définition** Soit deux vecteurs de caractéristiques  $V_1 = (H_1, H_2, H_3, \dots, H_{N_b^2})$  et  $V_2 = (G_1, G_2, G_3, \dots, G_{N_b^2})$  générés par ce descripteur sur deux images avec  $H_k$  (resp.  $G_k$ ) l'histogramme du bloc  $k$  de l'image 1 (resp. image 2).

Pour calculer la similarité  $\text{sim}(V_1, V_2)$  entre  $V_1$  et  $V_2$ , on commence par normaliser les histogrammes des blocs de chacun des deux vecteurs pour obtenir  $W_1 = (h_1, h_2, h_3, \dots, h_{N_b^2})$  et  $W_2 = (g_1, g_2, g_3, \dots, g_{N_b^2})$ .

On somme ensuite les divergences de Jeffrey[9] entre ces histogrammes normalisés des deux vecteurs. Cette somme est ensuite normalisée par le nombre de blocs  $N_b \times N_b$  comme l'indique l'équation 1.

$$\text{sim}(V_1, V_2) = \sum_{n=1}^{N_b \times N_b} D_{\text{jeff}}(h_n, g_n) / (N_b \times N_b) \quad (1)$$

La divergence de Jeffrey  $D_{\text{jeff}}$  est la version symétrique de la divergence de Kullback-Leibler  $D_{kl}$  qui est définie dans l'équation 2 [9]. Ainsi, pour deux distributions de probabilité discrètes  $P$  et  $Q$  (c'est-à-dire deux histogrammes), la distance de Kullback-Leibler entre  $P$  et  $Q$  est égale à

$$D_{kl}(P \parallel Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right) \quad (2)$$

**Utilisation de l'implémentation proposée** : Une classe **Similarity** comportant des méthodes statiques a été mise en place pour les différentes mesures de similarité. Elle comporte notamment la méthode **blockHistSim** qui implémente la mesure de similarité définie dans l'équation 1. L'appel à cette méthode est réalisé en ligne 12 sur les deux vecteurs de caractéristiques **V1** et **V2** issues des images **image1** et **image2** (lignes 10 et 11)

```
1 #include "Extractor.h"
2
3 int nbl = 3;
```

```

4  int nbn    = 10;
5  BlockHistDescriptor myBHD(nbl, nbn);
6  Mat  image1, image2;
7  image1 = imread(argv[1], 0);
8  image2 = imread(argv[2], 0);
9  vector<int> V1, V2;
10 myBHD.calcFeatures(image1, V1);
11 myBHD.calcFeatures(image2, V2);
12 double sim = Similarity::blockHistSim(V1, V2, nbl, nbn);

```

### 2.1.4 Validation de l'implémentation

La validation des résultats obtenus s'est faite en comparant les vecteurs de caractéristiques produit par notre classe avec ceux obtenus par le code Matlab existant, sur une base de données simple comportant 15 images. La mesure de similarité permet d'effectuer les comparaisons.

**Code de test** : Dans cet exemple, on charge une base de données produite par le code existant dans deux objets de la classe **DscDataBase**, puis on applique notre descripteur sur l'une d'elles avec la méthode **calcDescriptor**. Après, on extrait la moyenne et la variance de la similarité entre les deux bases par le routine **meanVarSimDataBase** de la classe **Similarity**.

```

1  #include "Extractor.h"
2
3  BlockHistDescriptor myDesc(3,10);
4  DscDataBase<int> mdbs1(argv[1]), mdbs2(argv[1]);
5  mdbs1.calcDescriptor(myDesc);
6  mean_var m_v = Similarity::meanVarSimDataBase(mdbs1, mdbs2,
7      Similarity::blockHistSim, 3, 10);
8  cout << "mean:" << m_v.mean << endl;
9  cout << "var:" << m_v.var << endl;

```

**Résultats du test** : La table 1 résume les résultats du test où la base de données utilisée comporte 15 images. En effet, pour chaque image de la base, une mesure de similarité a été calculée entre le vecteur de caractéristiques issu de l'implémentation Matlab et le vecteur issu de la bibliothèque C++ proposée.

La table 1 indique donc moyenne et variance de ces 15 mesures. La moyenne et la variance des similarités obtenues sont de l'ordre de  $10^{-3}$  ce qui valide notre implémentation pour cet extracteur.

Moyenne	Variance
0.921e-3	0.367e-3

TABLE 1 – Statistiques des similarités obtenues entre les vecteurs des caractéristiques obtenus via Matlab et via C++, pour le descripteur **Block-Histogram** sur une base de données de 15 images

## 2.2 Extracteur Angular radial partitioning (ARP)

Cet algorithme d'extraction s'intéresse particulièrement aux contours de l'image tout en évitant la segmentation d'objet. En effet, il est basé sur l'accumulation des points de contours dans des secteurs de l'image définis par un partitionnement radial et angulaire. L'approche utilise l'amplitude de la transformation de Fourier pour permettre une invariance par rotation. Comme cité dans [10], cette méthode est invariante aux transformations comme le changement d'échelle, la rotation et tolérante à de faibles transformations par translation ou érosion.

Les caractéristiques extraites ont l'avantage d'être compactes, rapides à extraire et à apparier. Par ailleurs, l'efficacité de l'ARP et le temps nécessaire à leur extraction sont comparables avec les quatre méthodes bien connues de la littérature qui se basent sur ANMRR<sup>4</sup>.

### 2.2.1 Algorithme de l'extracteur

L'objectif principal de l'ARP est de transformer l'image en une nouvelle structure qui supporte une mesure de similarité entre images de manière facile et efficace, tout en capturant des propriétés invariantes par changement d'échelle et rotation. La carte des contours d'une image porte en elle une structure solide de l'image, indépendamment de ses attributs de couleurs. Son utilisation est très fréquente dans les domaines du traitement d'images, de reconnaissance de formes et de récupération d'images [10].

**Principe de l'algorithme** Dans cette approche, on commence par convertir les images de la base de données en niveaux de gris puis on leur applique un filtre de Canny [11] pour récupérer les images de contours. Pour réaliser l'invariance d'échelle, les images de contours résultantes sont normalisées en des images de  $\text{dim} \times \text{dim}$  pixels, de façon à insérer l'image de

---

4. the average normalised modified retrieval rank

contours dans une image carrée dont la dimension est en général,  $\text{dim} = \max(\text{image.width}, \text{image.height})$ . L'image est ensuite recadrée dans une grille  $[-1, 1] \times [-1, 1]$  incluse dans le cercle unité de façon à associer à chaque pixel de l'image, le centre de sa case correspondante dans la grille. Cette image normalisée sera notée  $I$  et va être utilisée pour extraire le vecteur de caractéristiques. Dans la suite, on considérera que le pixel  $I(\rho, \theta)$  est égale soit à '1' pour les pixels de contour ou '0' si non.

Pour extraire les caractéristiques, on considère que le cercle unité dans lequel l'image est incluse, comme un histogramme pour lequel les bins sont les sections radiales-angulaires. La figure 4 décrit la géométrie de ces sections radiales-angulaires (ou bins). Ainsi, on somme dans chaque bin, le nombre de pixels de contour tombant dans la section correspondante. L'histogramme final obtenu dit de contours, a pour taille : le nombre  $\text{arpRadial}$  de sections radiales  $\times$  le nombre de sections angulaires  $\text{arpAngular}$ .

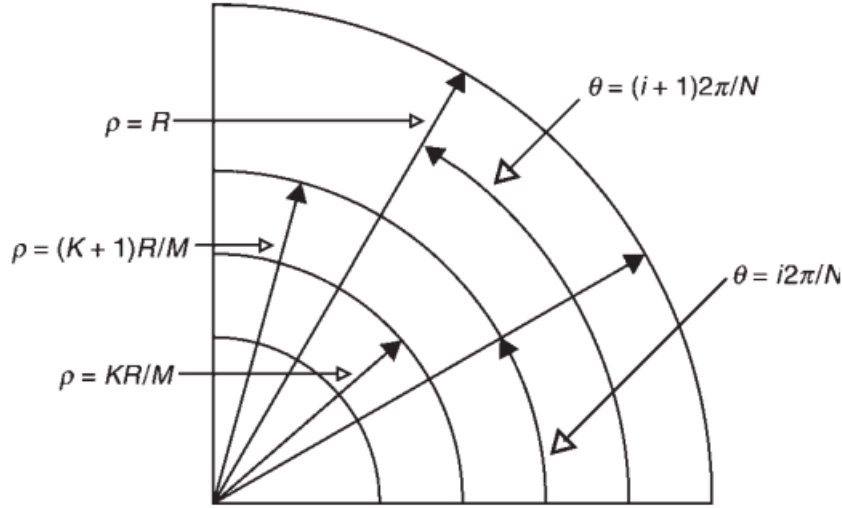


FIGURE 4 – Définition des sections radiales-angulaires du cercle unité.

**Description formelle** Pour la suite de ce paragraphe, on adopte les notations suivantes :

- $D = \sqrt{2} \cdot \text{dim}$ ,
- $N = \text{arpRadial}$ , le nombre de sections radiales,
- $M = \text{arpAngular}$ , le nombre de sections angulaires.

Avec des indices  $i$  et  $j$  de pixels qui vont donc varier entre 0 et  $\dim - 1$ , les coordonnées cartésiennes  $X_i$  et  $Y_j$  dans le cercle unité s'expriment par

$$X_i = (2 * i + 1 - \dim) / D \quad (3)$$

$$Y_j = (2 * j + 1 - \dim) / D \quad (4)$$

Les coordonnées polaires  $\rho_{i,j}$  et  $\theta_{i,j}$  dans le cercle unité deviennent alors

$$\rho_{i,j} = \sqrt{X_i^2 + Y_j^2} \quad (5)$$

$$\theta_{i,j} = \text{atan2}(Y_j, X_i) \quad (6)$$

Le descripteur  $V_c$  produit par cet algorithme d'extraction est alors de taille  $M \times N$ , et contient l'ensemble des valeurs  $f_{k,l}$  avec l'agencement suivant

$$V_c = (f_{1,1}, \dots, f_{1,M}, \dots, f_{k,1}, \dots, f_{k,M}, \dots, f_{N,1}, \dots, f_{N,M}) \quad (7)$$

où  $k \in [0, N - 1]$  et  $l \in [0, M - 1]$ .

Chaque composant  $f_{k,l}$  de ce vecteur de caractéristiques est défini par

$$f_{k,l} = \sum_{\rho=\frac{k}{N}}^{\frac{k+1}{N}} \sum_{\theta=\frac{l2\pi}{M}}^{\frac{(l+1)2\pi}{M}} I(\rho, \theta) \quad (8)$$

### 2.2.2 Utilisation de la bibliothèque

La bibliothèque **Extractor** a une classe nommée **ArpDescriptor** dont le mode de fonctionnement est similaire à celui de l'extracteur précédent, comme le montre le listing suivant.

```
1 #include "Extractor.h"
2
3 Mat img = imread(argv[1], 0);
4 vector<int> Vc;
5 int arpRadial = 4, arpAngular = 8;
6 ArpDescriptor marp( arpRadial, arpAngular, img.size());
7 marp.calcFeatures(image, Vc);
```

Le constructeur de cet objet prend comme paramètres le nombre de sections radiales, le nombre de sections angulaires et la taille des images de la base de données (ligne 6).



### 2.2.3 Mesure de similarité

Le descripteur produit par cet algorithme d'extraction étant un histogramme, la mesure de similarité est réalisée de la même manière que l'extracteur **BlockHistogram**.

```
1 #include "Extractor.h"
2
3 Mat image1, image2;
4 image1 = imread(argv[1], 0);
5 image2 = imread(argv[2], 0);
6 int arpRadial = 4, arpAngular = 8;
7 ArpDescriptor marp( arpRadial, arpAngular, image1.size());
8 vector<int> V1, V2;
9 marp.calcFeatures(image1, V1);
10 marp.calcFeatures(image2, V2);
11 cout << "symilarity:" << Similarity::histSim(V1, V2);
```

L'utilisation de la bibliothèque à ce propos consiste à appeler la méthode **histSim** de la classe **Similarity** en lui passant deux vecteurs produits par ce même extracteur (lignes 9 à 11).

### 2.2.4 Validation de l'implémentation

**Code du test** On retrouve une procédure similaire à celle de l'extracteur précédent.

```
1 #include "Extractor.h"
2
3 // pour une base d'images de dimension (240,320)
4 int dbsImgW = 240, dbsImgH = 320;
5
6 // on initialise un descripteur de type ARP
7 ArpDescriptor myDesc( 4 , 8 , Size(dbsImgW, dbsImgH));
8
9 // On charge la base dans deux objets
10 DscDataBase<int> mdbs1(argv[1]), mdbs2(argv[1]);
11
12 // On applique notre descripteur sur l'une d'elles
13 mdbs1.calcDescriptor(myDesc);
14
15 // On calcule la moyenne et la variance
16 mean_var m_v = Similarity::meanVarSimDataBase(mdbs1, mdbs2,
17 Similarity::histSim);
18 cout << "mean:" << m_v.mean << endl;
19 cout << "var:" << m_v.var << endl;
```

Moyenne	Variance
0.066	0.063

TABLE 2 – Statistiques des similarités obtenues entre les vecteurs des caractéristiques obtenus via Matlab et via C++, pour le descripteur **ARP** sur une base de données de 15 images

**Résultats du test** La table 2 résume les résultats du test où la base de données utilisée comporte 15 images. En effet, pour chaque image de la base, une mesure de similarité a été calculée entre le vecteur de caractéristiques issu de l’implémentation Matlab et le vecteur issu de la bibliothèque C++ proposée.

La table 2 indique donc moyenne et variance des mesures de similarité sur ces 15 mesures. La moyenne de 0.066 et la variance de 0.063 sont acceptables. Ces différences (plus importantes que dans le cas précédent) viennent principalement du fait que le filtre de Canny implémenté dans Matlab est légèrement différent de celui de OpenCV.

### 3 Descripteurs basés sur les moments

La forme est généralement une description riche et souvent très caractéristique d’un objet. L’extraction de caractéristiques géométriques a été le fer de lance de la recherche d’image par le contenu (CBIR) ces dernières années [12]. Dans cette seconde partie, on va considérer des descripteurs de formes qui exploitent la notion de moments invariants pour caractériser l’intégralité de la forme d’une région. Ces attributs de forme sont robustes aux transformations géométriques comme la translation, la rotation et le changement d’échelle.

Les moments géométriques d’ordre  $p + q$  d’une fonction  $f$  sont définis par l’équation (9).

$$m_{p,q} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x^p y^q f(x, y) dx dy \quad (9)$$

Il est démontré que, pour les fonctions qui n’ont des valeurs non nulles que sur une région bornée de l’espace, tous les moments de cette fonction existent et permettent inversement de reconstruire complètement le fonction  $f$  [13].

Par exemple, le moment  $m_{0,0}$  d’ordre 0 représente l’aire dans l’espace occupé par la forme d’un objet. Les deux moments  $m_{0,1}$  et  $m_{1,0}$  d’ordre

1, ajoutés au moment d'ordre 0 permettent d'obtenir le centre de masse  $C(x_c, y_c)$  d'un objet.

$$x_c = \frac{m_{1,0}}{m_{0,0}} \quad (10)$$

$$y_c = \frac{m_{0,1}}{m_{0,0}} \quad (11)$$

### 3.1 Descripteur Moments de Hu (HuMomDescriptor)

A partir des coordonnées du centre de masse de l'objet, on peut définir ce qu'on appelle les moments centrés  $\mu_{pq}$  pour une image  $f$  de taille  $N \times M$  :

$$\mu_{pq} = \sum_{x=1}^N \sum_{y=1}^M (x - x_c)^p (y - y_c)^q f(x, y) \quad (12)$$

Ces moments centrés ont la propriété d'être invariants par translation.

On peut ainsi continuer de faire des transformations entre les moments des différents ordres et d'extraire plus d'information sur la forme de l'objet. Dans la suite on considérera la version normalisée  $\eta_{pq}$  de ces moments centrés.

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}} \quad (13)$$

Hu [13] a proposé plusieurs transformations non linéaires définies sur les moments géométriques normalisés  $\eta_{pq}$ , pour produire 7 moments qui sont invariants par translation, rotation et par changement d'échelles. Ces descripteurs ont été appliqués avec succès à l'identification d'avions, de navires et de visages.

Toutefois l'utilisation des moments présente certains points faibles comme leur gourmandise en temps de calcul et le fait qu'ils contiennent de la redondance d'information du fait qu'ils ne sont pas indépendants les uns des autres.

#### 3.1.1 Algorithme de l'extracteur

Ainsi, Hu a dérivé 7 moments à partir des moments centrés normalisés  $\eta_{pq}$ . Ces 7 moments sont donnés par les équations suivantes :

$$M_1 = \eta_{20} + \eta_{02}$$

$$M_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

$$M_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$M_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$M_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

$$M_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$

$$M_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

Par conséquent, l'algorithme d'extraction consiste à calculer à partir de l'image, les moments centrés et normalisés puis à en dériver les 7 moments de Hu qui formeront le vecteur de caractéristiques du descripteur  $V_c = (M_1, \dots, M_7)$ .

### 3.1.2 Utilisation de la bibliothèque

La bibliothèque **Extractor** contient la classe **HuMomDescriptor** qui permet de manière similaire aux autres extracteurs, d'extraire les moments de Hu d'une image par le biais de la méthode **calcFeatures**. Le constructeur par défaut du module ne prend pas de paramètres et dans ce cas l'image est considérée comme étant une image binaire.

```
1 #include "Extractor.h"
2
3 Mat img = imread(argv[1], 0);
4 vector<double> Vc;
5 HuMomDescriptor mhmDesc;
6 mhmDesc.calcFeatures(image, Vc);
```

Quand on envoie au constructeur du descripteur comme paramètre **GRAY** (resp. **EDGES**) l'image est transformée en niveau de gris (resp. carte de contours). Par défaut l'image est binarisée avant l'extraction des moments.

### 3.1.3 Mesure de similarité

La mesure de similarité implémentée pour ce descripteur est la distance de Mahalanobis.

**Principe** La distance de Mahalanobis est utilisée pour la classification basée sur les différences statistiques. Cette mesure est un outil très utile pour déterminer la similarité entre un vecteur de caractéristiques extrait d'un échantillon inconnu et un ensemble de caractéristiques extrait d'une classe d'échantillons connus. Elle est donnée par l'équation suivante :

$$D^2 = (V_c - \nu)' C^{-1} (V_c - \nu) \quad (14)$$

où  $D$  est la distance de Mahalanobis entre le vecteur de caractéristiques  $V_c$  et un ensemble de caractéristiques extrait d'une classe d'échantillons connus avec  $\nu$  la moyenne de ces vecteurs caractéristiques et  $C$ , leur matrice de covariance.

**Utilisation** : La distance de Mahalanobis est implémenté dans la classe **Similarity** sous le nom **mahalanobisDist**. Elle prend en paramètres le vecteur de caractéristique obtenu par le descripteur (ligne 6) et la base contenant les descripteurs des échantillons (ligne 4).

```
1 #include "Extractor.h"
2
3 HuMomDescriptor myDesc;
4 DscDataBase<double> mdbs(argv[1]);
5 vector<double> V;
6 myDesc.calcFeatures(image, V);
7 double sim = Similarity::mahalanobisDist(V, mdbs);
8 cout <<"similarity:" << sim <<endl;
```

### 3.1.4 Validation de l'implémentation

L'implémentation de cet extracteur utilise des routines déjà existantes dans OpenCV. Les résultats obtenus sont différents de ceux produits par le code de référence Matlab. On a donc adopté une démarche expérimentale et qualitative pour valider les résultats obtenus par l'implémentation C++ proposée.

**Vérification de la décroissance des moments d'ordre** De par leur expression mathématique, les moments de Hu doivent diminuer alors que l'ordre augmente. L'objectif du premier test est de vérifier cette décroissance sur des images de test. La figure 5 montre un exemple d'images de test sur laquelle les moments de Hu sont calculés. Ces moments sont indiqués dans la table 3 où on peut effectivement observer la décroissance des valeurs de ces moments alors que l'ordre des moments augmente.



FIGURE 5 – Image binaire de test représentant une chauve-souris

$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$
0.3213	0.0396	0.0202	0.0015	4.949e-08	-7.276e-05	8.892e-06

TABLE 3 – Moments de Hu calculés pour l'image de test en figure 5.

Ce premier test permet de valider que l'ordre de grandeur des moments de Hu obtenu par notre algorithme varie conformément avec ce qu'on doit obtenir théoriquement.

**Vérification de l'invariance par rotation** La table 4 indique les moments de Hu obtenus pour une forme ovale avec plusieurs orientations.

Les moments obtenus sont presque égaux ce qui valide la fait que notre algorithme extrait bien un descripteur invariant par rotation.

**Vérification du caractère discriminant du descripteur implémenté**

Pour ce test, on a extrait les vecteurs de caractéristiques d'une base de données de 15 images binaires représentant des chauves-souris. Ensuite, les distances de Mahalanobis ont été calculées entre cette base de chauves-souris et




Image	$M_1$	$M_2$	$M_3$	$M_4$	$M_5$	$M_6$	$M_7$
	0.212	4.464e-06	1.864e-09	9.494e-08	1.363e-16	1.669e-11	1.256e-15
	0.211	4.104e-06	1.810e-09	9.159e-08	1.196e-16	1.220e-11	1.173e-15
	0.212	4.464e-06	1.864e-09	9.494e-08	1.363e-16	1.669e-11	1.256e-15

TABLE 4 – Moments de Hu pour une forme ovale sous différentes rotations






					
Mahalanobis distance	1.8909	1.8811	1.2988	1.2881	3.3982
Moyenne	1.95				
Variance	0.77				

TABLE 5 – Distances de Mahalanobis entre une base de chauves-souris et d'autres images de chauves-souris

- 5 images de la même classe, c'est-à-dire représentant des chauves-souris,
  - 5 images d'une autre classe, c'est-à-dire représentant des pommes.
- Les tables 5 et 6 donnent les distances de Mahalanobis obtenues.






					
Mahalanobis distance	4.5919	4.6137	4.6652	4.585	4.5942
Moyenne	4.61012				
Variance	0.0291545				

TABLE 6 – Distances de Mahalanobis entre une base de chauves-souris et des images de pommes

D’après les moyennes & variances des tables 5 et 6, on peut voir que si, dans ce cas de test simple, on fixe empiriquement 4 comme seuil de similarité, on peut discriminer correctement les chauves-souris des pommes. En effet, on observe que les distances entre les différentes chauves-souris sont voisines de 1.95 alors que celles entre chauves-souris et pommes dépassent 4.5.

Ces différents tests nous ont permis de valider que le descripteur implémenté a les caractéristiques d’invariance et de discrimination attendu théoriquement de moments de Hu.

### 3.2 Extracteur Angular Radial Transformation (ART)

Les descripteurs basés sur les moments souffrent de la redondance d’informations contenue dans les caractéristiques extraites. L’une des approches pour palier à cette problématique est de décomposer l’image dans une base orthogonale de façon à ce que les caractéristiques extraites soient indépendantes les unes des autres.

L’ART<sup>5</sup> ou transformation angulaire radiale [14] décompose les informations de l’image (niveaux de gris, contours ou sections binaires) dans une base de fonctions complexes orthogonales définies sur le disque unité. Le résultat est un ensemble de moments complexes de taille variable (dépendant du nombre de fonctions de la base prise en compte) et indépendants les uns des autres.

La base de fonctions  $V_{n,m}(\rho, \theta)$  considérées par l’ART, d’ordres  $n$  et  $m$ ,

---

5. Angular Radial Transformation



est définie sur le disque unité par l'équation (18).

$$V_{n,m}(\rho, \theta) = \frac{1}{2\pi} \exp(jm\theta) R_n(\rho) \quad (15)$$

où  $R_n(\rho)$  s'exprime selon l'expression suivante :

$$R_n(\rho) = \begin{cases} 1, & \text{si } n = 0 \\ 2 \cos(2\pi\rho), & \text{sinon} \end{cases} \quad (16)$$

Les moments  $F_{n,m}$  d'ordres  $n$  et  $m$  d'une image sont ensuite calculés par l'équation (18).

$$F_{n,m} = \langle V_{nm}, f \rangle \quad (17)$$

$$= \int_0^{2\pi} \int_0^1 V_{n,m}^*(\rho, \theta) f(\rho, \theta) \rho d\rho d\theta \quad (18)$$

où  $f$  est la fonction intensité normalisée de l'image en coordonnées polaires et  $V_{n,m}$  est la fonction d'ordres  $n$  et  $m$  de la base.

### 3.2.1 Algorithme de l'extracteur

**Principe général** Dans cette algorithme, on commence d'abord par normaliser l'intensité de l'image  $I$  entre 0 et 1. Ensuite, l'image de taille  $\text{dim} \times \text{dim}$  sur une grille  $[-1, 1] \times [-1, 1]$  (comme dans l'ARP). Les fonctions  $V_{n,m}(\rho, \theta)$  de la base sur le cercle intérieure de la grille sont ensuite calculées pour décomposer l'image dans cette base.

**Description formelle** Pour la suite de ce paragraphe, on note :

- $D = \text{dim}$ ,
- $DxDy = 4/\text{dim}^2$ ,

Avec des indices  $i$  et  $j$  de pixels qui vont donc varier entre 0 et  $\text{dim} - 1$ , les coordonnées cartésiennes  $X_i$  et  $Y_j$  dans le cercle unité s'expriment par

$$X_i = (2 * i + 1 - \text{dim})/D \quad (19)$$

$$Y_j = (2 * j + 1 - \text{dim})/D \quad (20)$$

Les coordonnées polaires  $\rho_{i,j}$  et  $\theta_{i,j}$  dans le cercle unité deviennent alors

$$\rho_{i,j} = \sqrt{X_i^2 + Y_j^2} \quad (21)$$

$$\theta_{i,j} = \text{atan2}(Y_i, X_j) \quad (22)$$

Le descripteur  $V_c$  produit par cet algorithme d'extraction est alors de taille  $M \times N$ , et contient l'ensemble des valeurs  $F_{n,m}$  avec l'agencement suivant

$$V_c = (F_{0,1}, \dots, F_{0,M}, \dots, F_{n,0}, \dots, F_{n,M}, \dots, F_{N,0}, \dots, F_{N,M}) \quad (23)$$

où  $n \in [0, N]$  et  $l \in [0, M]$ .

Chaque composant  $F_{n,m}$  de ce vecteur de caractéristiques est défini par

$$F_{n,m} \simeq \sum_{\rho_{i,j} \leq 1} I(i, j) V_{n,m}^*(\rho_{i,j}, \theta_{i,j}) DxDy \quad (24)$$

### 3.2.2 Utilisation de la bibliothèque

La bibliothèque **Extractor** contient la classe **ArtDescriptor** qui permet d'extraire le vecteur des caractéristiques d'une image avec la méthode **calcFeatures**.

Les paramètres du descripteur sont :

- $N = \text{artRadial}$  et  $M = \text{artAngular}$ , les ordres radial et angulaire maximum des fonctions de la base,
- la taille des images de la base (pour pouvoir générer le mapping entre coordonnées cartésiennes et polaires,
- un flag facultatif **Flag** qui par défaut vaut **EDGES**, permet de spécifier que l'image à traiter doit d'abord être transformée en une image de contours.

```

1 #include "Extractor.h"
2
3 Mat img = imread(argv[1], 0);
4 ArtDescriptor artDes(5, 5, image.size());
5 //On peut ajouter un flag éventuellement pour
6 //traiter l'image en binaire(BINARY) ou en niveaux de gris(GRAY)
7
8 vector<complex<double>> features ;
9 artDes.calcFeatures(image, features);

```

Dans cet exemple, on charge une image dans la matrice **img** via OpenCV (ligne 3). On crée ensuite un objet de type **ArtDescriptor** avec **artRadial** = 5 et **artAngular** = 5 (ligne 4). Enfin, on demande l'extraction du vecteur des caractéristiques qui dans le cas de l'ART, est un vecteur complexe (ligne 5).

On observe une interface de programmation similaire aux descripteurs précédents.

### 3.2.3 Mesure de similarité

La mesure de similarité est définie par la norme  $\mathcal{L}_1$  appliquée sur le vecteur des différences entre vecteurs de caractéristiques.

Il suffit d'invoquer la méthode **nomL1** de **Similarity** en passant les deux vecteurs à comparer, en paramètres.

```
1 #include "Extractor.h"
2
3 //On charge deux images dans des matrices
4 Mat img1 = imread(argv[1],0);
5 Mat img2 = imread(argv[2],0)
6
7 //On initialise un descripteur type arp 5x5
8 ArtDescriptor artDes(5,5, image.size() );
9
10 vector<complex<double> > v1,v2 ;
11 artDes.calcFeatures(image, v1);
12 artDes.calcFeatures(image, v2);
13
14 cout <<"sim:"<<Similarity::normL1<complex<double> >(v1,v2);
```

Dans cet exemple de test, on charge deux images dans deux matrices `img1` et `img2`. ensuite on initialise deux vecteur complexes puis on les remplit par les vecteurs caractéristiques de `img1` et `img2`. Enfin on fait appel à la routine `normL1` de **Similarity** en indiquant le type des vecteurs (dans ce cas `complex`).

### 3.2.4 Validation de l'implémentation

La validation des résultats est faite de manière similaire à la validation du descripteur ARP en comparant les deux bases (de 15 images) : celle générée par le code Matlab et celle générée par notre implémentation en C++.

```
1 #include "Extractor.h"
2
3 //on charge les deux bases celle produite
4 //par matlab et celle produite par notre classe
5 DscDataBase<complex<double> > mdbs1(argv[1]), mdbs2(argv[2]);
6
7 //on fait appel au module Similarity pour
8 //extraire la moyenne et la variance entre les deux bases
9 mean_var m_v = Similarity::meanVarSimDataBase(mdbs1, mdbs2,
```

Moyenne	Variance
0.648492	0.499749

TABLE 7 – Statistiques des similarités obtenues entre les vecteurs des caractéristiques obtenus via Matlab et via C++, pour le descripteur **ART** sur une base de données de 15 images

```

10 Similarity::normL1<complex<double> >);
11 cout << "mean:" << m.v.mean <<endl;
12 cout << "var:" << m.v.var <<endl;

```

Dans cet exemple, on charge (depuis des fichiers csv) dans `mdbs1` les vecteurs produit par notre implémentation et dans `mdb2` ceux produit par le code Matlab (ligne 5). On fait ensuite appelle à la fonction **meanVarSimDataBase** (ligne 9-10) pour calculer la moyenne et la variance de la norme L1 entre les vecteurs des deux bases.

**Résultats du test de validation** La table 7 indique donc moyenne et variance sur les 15 mesures de similarité calculées. La moyenne de similarité entre nos résultats et ceux de Matlab pour cette implémentation est de 0.65 tandis que la variance est 0.5, ce qui est tolérable. Cette faible différence provient de la différence entre les routines Matlab et C++, qui génèrent l'image des contours.

### 3.3 Extracteur Moments de Zernike

Les moments de Zernike sont souvent utilisés pour extraire les caractéristiques globales d'une image dans les domaines de reconnaissance et d'analyse d'images. Ils ont été introduits pour la première fois dans le domaine de la vision par ordinateur par Teague [15]. Ce descripteur s'est révélé supérieur aux autres fonctions issues des moments de par sa capacité de description et sa robustesse au bruit et aux déformations. De nombreux chercheurs se sont penchés sur ces moments de Zernike, principalement pour optimiser leur temps de calcul et *améliorer* leur précision.

#### 3.3.1 Principe de calcul des moments de Zernike

**Polynômes de Zernike** Concrètement, un moment de Zernike est un nombre complexe. Comme pour l'ART, le principe est de décomposer l'image

dans une base orthogonale. Pour ce descripteur, la base retenue est formée par les polynômes de Zernike ( $Z_m^n$ ). Il s'agit d'un ensemble de polynômes complexes définis sur le cercle unité par l'équation (25).

$$Z_m^n(\rho, \phi) = R_m^n(\rho) \exp(jm\phi) \quad (25)$$

où  $m$  et  $n$  sont des entiers avec  $|m| \leq n$  et  $(n - |m|)$  pair. On notera  $\Lambda$  l'ensemble des entiers vérifiant ces conditions.

Dans ce cas,  $R_m^n(\rho)$  est donnée par l'équation (26).

$$R_m^n(\rho) = \sum_{k=0}^{\frac{n-|m|}{2}} \frac{(-1)^k (n-k)!}{k! \left(\frac{n+|m|}{2} - k\right)! \left(\frac{n-|m|}{2} - k\right)!} \rho^{n-2k} \quad (26)$$

Les polynômes de Zernike ont la propriété d'être inclus dans le disque unité, c'est-à-dire que  $|Z_m^n(\rho, \phi)| \leq 1$ .

**Moments de Zernike** Les moments de Zernike  $F_{n,m}$  d'ordre  $n$  et de répétition  $m$  d'une fonction  $f$  définie sur le disque unité, sont donnés par l'équation (28).

$$F_{n,m} = \langle Z_m^n, f \rangle \quad (27)$$

$$= \int_0^{2\pi} \int_0^1 (Z_m^n(\rho, \theta))^* f(\rho, \theta) \rho d\rho d\theta \quad (28)$$

Le modules des moments de Zernike sont invariants par rotation, c'est pour cela qu'en général on les prend comme descripteurs des images. L'orthogonalité de la base des polynômes de Zernike permet de reconstruire l'image à partir de ces moments. Le nombre de moments suffisant pour avoir une reconstruction satisfaisante de l'image nous permet de déterminer empiriquement jusqu'à quel ordre on peut se limiter pour avoir une description suffisante des images étudiées.

### 3.3.2 Algorithme de l'extracteur

**Principe général** Dans cet algorithme, on commence d'abord par normaliser l'image de façon à l'insérer au centre d'une image carrée de dimensions  $\text{dim} \times \text{dim}$ . Cette image est ensuite mappée (comme pour l'ARP) dans une grille  $[-1, 1] \times [-1, 1]$ .

Les fonctions  $R_n^m$  de la base de Zernike (jusqu'à l'ordre maximal passé comme paramètre) sur le cercle intérieur de la grille sont ensuite estimées afin que l'image puisse être décomposée dans la base des polynômes  $Z_m^n$ . Pour calculer les fonctions  $R_n^m$  de la base, on génère à l'avance la liste des indices et la liste des quotients (de factorielles) dont on aura besoin pour calculer les polynômes.

**Description formelle** Dans la suite du paragraphe, on considère les conventions suivantes :

- $D = \text{dim}$ ,
- $DxDy = 4/\text{dim}^2$ ,
- $N = \text{maxOrder}$ ,
- $\Lambda = \{(n, m) \in (\mathbb{N} \times \mathbb{Z}), |m| \leq n \wedge (n - |m|) \text{ est paire}\}$ .

Avec des indices  $i$  et  $j$  de pixels qui vont donc varier entre 0 et  $\text{dim} - 1$ , les coordonnées cartésiennes  $X_i$  et  $Y_j$  dans le cercle unité s'expriment par

$$X_i = (2 * i + 1 - \text{dim})/D \quad (29)$$

$$Y_j = (2 * j + 1 - \text{dim})/D \quad (30)$$

Les coordonnées polaires  $\rho_{i,j}$  et  $\theta_{i,j}$  dans le cercle unité deviennent alors

$$\rho_{i,j} = \sqrt{X_i^2 + Y_j^2} \quad (31)$$

$$\theta_{i,j} = \text{atan2}(Y_i, X_j) \quad (32)$$

On associe à chaque couple  $(m, n) \in \Lambda$ , un indice  $k$  qui sert d'index à 2 listes `nIndex` et `mIndex`, ces listes lient chaque indice  $k$  avec le couple correspondant de façon à avoir `nIndex[k] = n` et `mIndex[k] = m` où  $k$  varie entre 0 et  $\text{card}(\Lambda)$ .

On génère ensuite la liste des quotients  $(Q)_{k,s}$  :

$$Q_{k,s} = \frac{(n - s)!}{s! \left(\frac{n+|m|}{2} - s\right)! \left(\frac{n-|m|}{2} - s\right)!} \quad (33)$$

avec  $0 \leq s \leq \frac{n-|m|}{2}$  et  $k$  tel que `nIndex[k] = n` et `mIndex[k] = m`.

Les polynômes de la base peuvent ensuite être calculés et stockés dans une liste  $(Z)_k$  où  $Z_k = Z_n^m$  et  $k$  tel que `nIndex[k] = n` et `mIndex[k] = m`.

L'image  $I$  est finalement décomposée dans la base des polynômes  $(Z)_k$  pour obtenir une liste de moments  $(F)_k$  qui sera retournée comme vecteur de caractéristiques :

$$F_k \simeq \frac{n+1}{\pi} \sum_{\rho_{i,j} \leq 1} I(i,j) Z_k^*(\rho_{i,j}, \theta_{i,j}) Dx Dy \quad (34)$$

### 3.3.3 Utilisation de la bibliothèque

Dans la bibliothèque, la classe **ZMDescriptor** permet par le biais de sa méthode **calcFeatures** d'extraire les moments de Zernike complexes, depuis une image donnée.

```
1 #include "Extractor.h"
2
3 Mat image = imread(argv[1], 0);
4 int order = atoi(argv[2]);
5 ZMDescriptor zmDes(order, image.size());
6 vector<complex<double>> features;
7 zmDes.calcFeatures(image, features);
```

le vecteur de caractéristiques doit être initialiser comme vecteur de complexes (ligne 6). L'ordre du descripteur désigne la valeur maximale de  $n$  dans  $\Lambda$  (ligne 4).

### 3.3.4 Mesure de similarité

La manière habituelle de comparer deux descripteurs  $Z$  et  $Z'$  de Zernike est une simple distance euclidienne entre les modules des moments, comme indiqué dans l'équation (35).

$$d(Z, Z')^2 = \sum_{(p,q) \in \Lambda} (|Z_{p,q}| - |Z'_{p,q}|)^2 \quad (35)$$

En rangeant les éléments de  $\Lambda$  dans une liste de façon à associer à chaque paire  $p, q$  un indice  $i$ , on peut normaliser et réécrire la mesure sous la forme suivante :

$$d(Z, Z')^2 = \frac{1}{N} \sum_{i \in \Lambda} (|Z_i| - |Z'_i|)^2 \quad (36)$$

### 3.3.5 Validation de l'implémentation

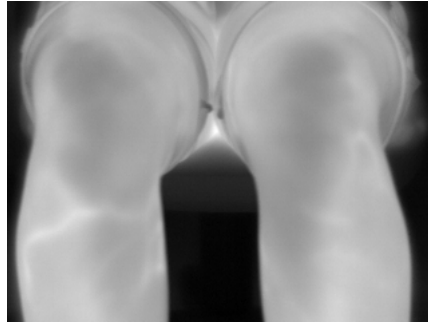
#### Comparaison entre résultat de Matlab et ceux de notre classe :

Les résultats déjà générés par le code Matlab pour ce descripteur diffèrent trop de ceux qu'on a obtenu par notre implémentation. la raison exacte derrière cette différence n'a pas pu être identifier donc comme pour le descripteur basé sur les moments de Hu on a fait des expérimentations pour valider nos résultats.

C++ résultats	0.487	-0.036+0.017i	-0.11	-0.157+0.005i
Matlab résultats	10947.314	523.683-668.538i	-13989.1342	2096.688+42.596i

TABLE 8 – Tableau illustrant la différence entre les quatre premiers moments de Zernike obtenu par notre code et ceux obtenu par le code matlab sur une image de test

FIGURE 6 – image de test



**validation de l'invariance par rotation :** Dans le tableau 9 on peut vérifier que notre implémentation extrait bien des moments dont le module est invariant par rotation.

**validation par reconstruction de l'image :** La validation de l'implémentation s'est faite aussi en examinant les résultats de reconstruction de l'image étudiée à partir des moment extraits. En effet, si les moments extraits sont corrects, il doivent en considérant un rang suffisamment grand, permettre de reconstruire l'image originale.

L'image sur laquelle ce test a été effectué est une image de Lena en niveaux de gris, de taille  $128 \times 128$  pixels. La table 10 présente cette image ainsi que






	$ Z_1 $	$ Z_2 $	$ Z_3 $	$ Z_4 $	$ Z_5 $	$ Z_6 $
	0.180669	0.00638368	0.328325	0.0665304	0.0152564	0.00705433
	0.180669	0.00638395	0.328319	0.0665324	0.0152542	0.00705404
	0.180669	0.00638303	0.328322	0.0665313	0.0152574	0.0070535

TABLE 9 – Premiers moments de zernike d’ordre 1 à 6 générés par notre implémentation pour une image sous différentes rotations

les images reconstruites à partir des moments de Zernike en ne considérant que les moments avec des ordres inférieurs à 20, 40, 50, 80 et 100.


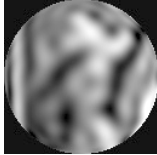




Image originale	ordre 20	ordre 40	ordre 50	ordre 80	ordre 100
					

TABLE 10 – Exemple de reconstruction d’une image de Lena à partir des moments de Zernike calculés via l’implémentation en C++.

On observe bien que la prise en compte d’un ensemble de moments plus grands permet une reconstruction plus fine de l’image initiale. Ce test permet de valider l’implémentation en C++, de l’extracteur des moments de Zernike.

## 4 Architecture de la bibliothèque

La bibliothèque **Extractor** comporte 4 modules et est dépendante de deux bibliothèques **OpenCV** v2++ et **GMP** v6++. Ces bibliothèques doivent être installées et intégrées dans le projet avant de pouvoir compiler la bibliothèque proposée.

Pour l’installation et l’intégration de ces deux bibliothèques, il faut se référer à leur manuel ou bien le site officiel pour plus de détails [16] pour OpenCV et [17] pour GMP.

## 4.1 Description des composants de la bibliothèque

**Interface Descriptor** Le premier composant de la bibliothèque **Extractor** est l'interface **Descriptor** (vide pour l'instant) dont hérite tous les descripteurs. Chaque sous-classe spécifique à un descripteur implémente au moins une méthode **calcFeatures** qui prend comme paramètres l'image source et le vecteur destination c'est-à-dire le vecteur où les caractéristiques calculées seront stockées. Ce vecteur pouvant être un vecteur d'entiers, de doubles, de complexes, ..., cette méthode **calcFeatures** ne possède pas la même signature pour l'ensemble des classes et ne peut donc faire l'objet d'une méthode virtuelle. Par ailleurs, certaines classes passent des paramètres supplémentaires à **calcFeatures**.

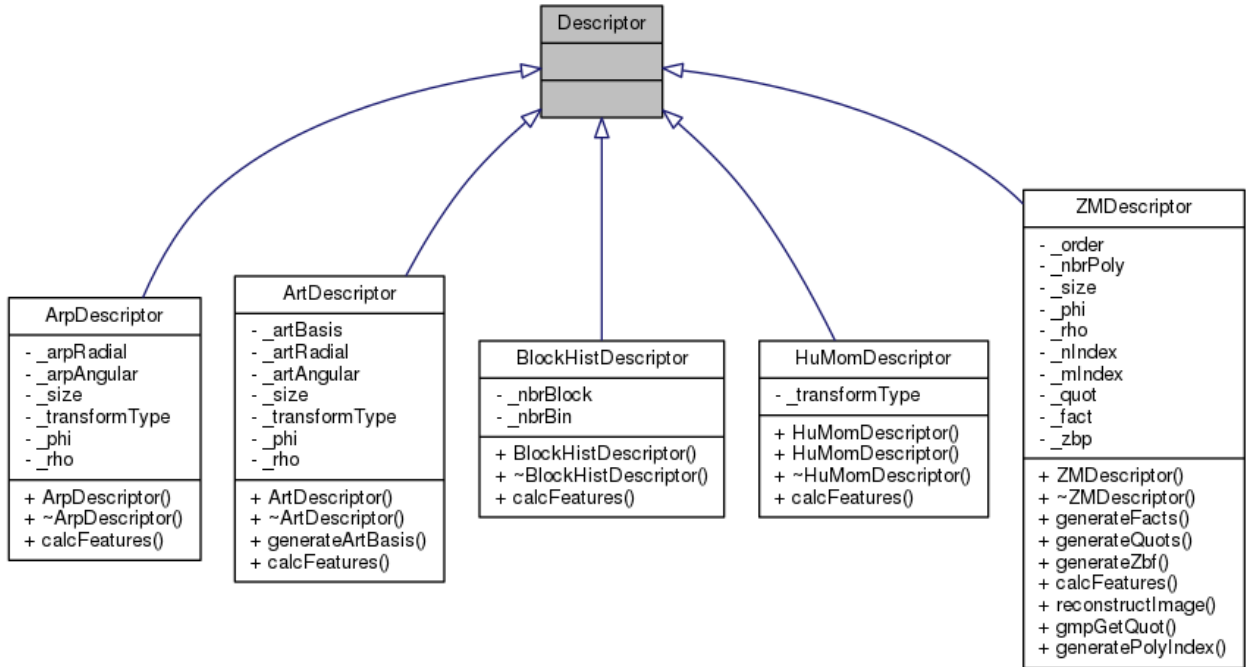


FIGURE 7 – Diagramme de classe pour la hiérarchie des classes de descripteurs.

Dans la figure 7, on retrouve bien les différentes classes vues tout au long du document.

**Classe Similarity** Le deuxième composant est la classe **Similarity**. Cette classe rassemble les différentes mesures de similarité qui s'appliquent soit sur deux vecteurs (ou bien deux bases de données) générés par le même algorithme d'extraction. Le choix de séparer les mesures de similarité des descripteurs a été fait pour apporter plus de flexibilité à l'architecture et aussi pour permettre le fait qu'un descripteur peut avoir plusieurs mesures de similarité différentes.

Cette classe permet aussi de faire la comparaison entre les descripteurs de deux bases de données en renvoyant la moyenne et la variance des mesures de similarité calculées. Ce point a été illustré dans les sections relatives à la validation des résultats de certains extracteurs. Cette classe contient des méthodes générales qui prennent en paramètres deux vecteur (de différents types produit par le même algorithme), une mesure de similarité et appliquent cette mesure aux deux vecteur.

#### Exemple de cas d'utilisation

```

1 #include "Extractor.h"
2
3
4 Mat img1 = imread(argv[1],0);
5 Mat img2 = imread(argv[2],0);
6 ArtDescriptor artDes(5,5, img1.size() );
7 vector<int> v1, v2;
8 artDes.calcFeatures(img1,v1);
9 artDes.calcFeatures(img1,v2);
10
11 double sim1=Similarity::getSim(v1,v2, Similarity::normL1<int>);
12 double sim2=Similarity::getSim(v1,v2, Similarity::normL2<int>);
13 double sim3=Similarity::getSim(v1,v2, Similarity::histSim);

```

Dans cet exemple on montre comment la bibliothèque **Similarity** peut permettre d'appliquer plusieurs mesures de similarité entre deux descripteurs (lignes 11,12,13).

**Classe Tools** Le troisième module est la classe **Tools** qui rassemble les routines de base qu'utilisent les différents algorithmes d'extracteur comme la méthode **getPolarCoordinates** qui permet de générer le mapping entre les indices des pixels de l'image et une grille en coordonnées polaires.

**Classe DscDataBase** Le quatrième composant est la classe **DscDataBase** qui a pour objectif de stocker les bases de descripteurs générées par les

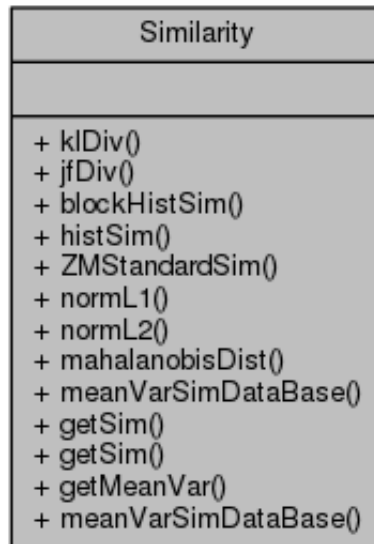


FIGURE 8 – Diagramme de classe pour la classe **Similarity**.

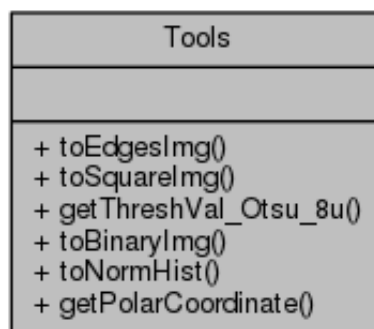


FIGURE 9 – Diagramme de classe pour la classe **Tools**.

différents algorithmes d'extraction. Chaque ligne d'une telle base de données comporte 4 champs :

- le nom de l'image,
- sa classe,
- son identifiant unique
- un vecteur de caractéristiques (d'un descripteur particulier).

Cette classe permet de travailler au niveau de la base de données en fournissant des routines de haut niveau :

- **loadDataBase** : Lecture d'un fichier `.csv` contenant une base de descripteurs, cette méthode est appelée par une surcharge du constructeur qui prends le path d'un fichier `.csv` comme paramètre.
- **calcDescriptor** : Extraction des vecteurs de caractéristique sur l'ensemble des images référencées dans la base,
- **loadFromFolder** : Chargement automatisé d'une liste d'images et extraction des vecteurs de caractéristiques correspondants en spécifiant le chemin d'accès au dossier contenant les images, l'extension des images à traiter et l'extracteur choisi.
- **save** : sauvegarde de la base de données dans un fichier `.csv`.
- **addRow** : ajouter une ligne à la base de données.

Des méthodes pour accéder au contenu spécifique d'une ligne de la base de données ont été implémentés. Pour plus de détails une documentation a été livrée avec la bibliothèque

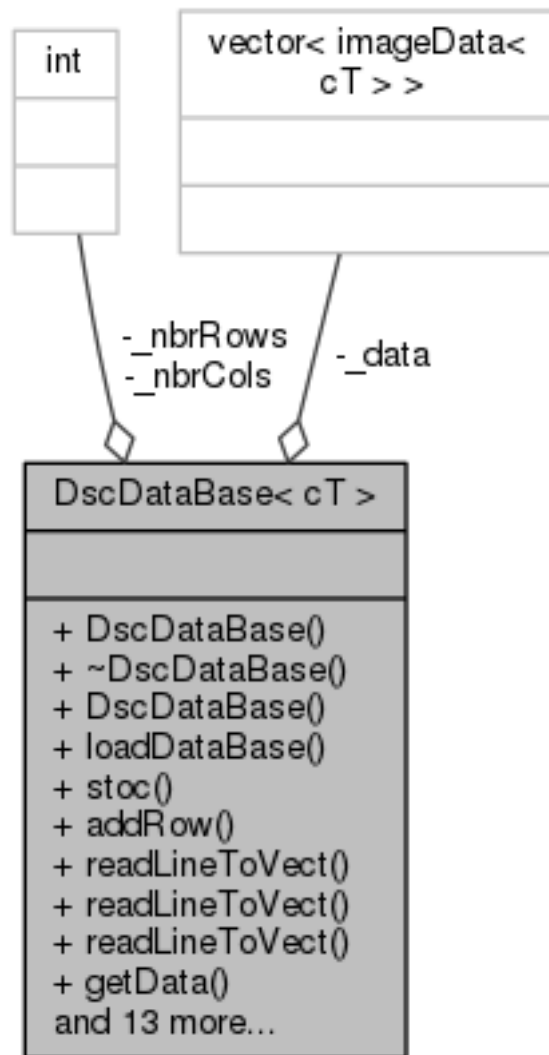


FIGURE 10 – Diagramme de classe pour la classe **DscDataBase**.

## 4.2 Réflexions sur l'architecture logicielle proposée

La bibliothèque a été codée de façon à ce qu'elle réponde à 3 critères à savoir la modularité, l'évolutivité et la facilité d'utilisation.

**modularité** : la bibliothèque est divisée en plusieurs modules chacun se chargeant de fonctionnalités particulières. Le module **Descriptor** se charge des différents algorithmes d'extraction, le module **Similarity** se charge de l'implémentation des mesures de similarité, le module **DscDataBase** se charge de la gestion des base de données de descripteurs générées par les algorithmes et le module **Tools** se charge des routines de base. Le choix de diviser la bibliothèque en de nombreux modules permet de bien l'organiser de façon à ce qu'il soit facile d'y appliquer des modification. Le fait de séparer les descripteurs des mesures de similarité est motivé par le fait qu'un descripteur peut bien avoir plus d'une mesure de similarité.

**évolutivité** : La bibliothèque a été architecturé de façon à ce quelle soit facile à intégrer et à développer d'avantage. Un développeur ou utilisateur futur peut facilement implémenter son propre descripteur et de l'intégrer dans la bibliothèque. Pour cela il suffit qu'il hérite de la classe **Descriptor** et qu'il implémente sa propre méthode **calcFeatures** recevant deux paramètres à savoir l'image et le vecteur de sortie. Un développeur peut aussi implémenter d'autres mesures de similarité adaptées à son cas et de les mettre dans la classe **Similarity**. En cas de besoin, d'autres routines qui agissent sur des images ou sur d'autres éléments, il suffit de les mettre dans classe **Tools** et les appeler depuis les autres modules.

**facilité d'utilisation** : La bibliothèque peut être facilement utilisée sans avoir besoin de comprendre comment sont codés les différents modules. Comme on l'a vu dans les sections sur les descripteurs, le mode d'emploi de la bibliothèque est assez générique et ne demande pas de syntaxe complexe. Par exemple pour utiliser n'importe quel descripteur la syntaxe à suivre est :

```
1 #include "Extractor.h"
2
3 Mat img = imread(argv[1],0);
4 (Descriptor) myDesc(parametre);
5 vector<(type)> myVect;
6 myDesc.calcFeatures(img,myVect)
```

où (Descriptor) est soit un `arpDescriptor` ou un `artDescriptor` ainsi de suite et le (type) est lié au type des composantes du vecteur de sortie du descripteur : ça peut être un entier par exemple pour les histogrammes ou un complexe...

## 5 Conclusion et perspective

Dans le cadre de ce stage, il était demandé d'implémenter en C++ une liste pré-établie d'extracteurs de descripteurs utilisés classiquement en CBIR et déjà codés en Matlab, les extracteurs étudiés dans ce stage peuvent être catégorisés en deux familles selon qu'elles se basent directement sur les histogrammes issus des images ou bien sur leurs moments. L'objectif principal de ces algorithmes est d'extraire des vecteurs de caractéristiques robustes à certaines transformations affectant les images comme des translations, des changements d'échelle ou des rotations. Il était aussi demandé de choisir et d'implémenter pour chacun de ces descripteurs, une mesure de similarité qui indique un degré de ressemblance entre vecteurs numériques d'un même descripteur. Grâce à cette mesure de similarité, il sera possible de valider la qualité des algorithmes codés en C++ par comparaison des vecteurs numériques produits avec ceux obtenus par les implémentations Matlab. Pendant ce stage nous avons implémenté l'ensemble de ces algorithmes d'extraction dans une bibliothèque modulaire, évolutive et facile à utiliser. Après chaque phase d'implémentation, une démarche a été suivie pour valider les résultats obtenus. Cette démarche consistait à faire une comparaison entre les bases de données générées par le code Matlab et celles générées par notre implémentation pour cela on a utilisé des mesures de similarité qu'on a choisi d'implémenter et qui sont adaptées à chaque type de descripteur. Dans certains cas, les résultats du code Matlab étaient très différents de ceux qu'on obtenait sans qu'on puisse identifier la cause derrière. Dans ces cas là on a adopté une démarche expérimentale pour valider que notre implémentation de ces algorithmes a les caractéristiques qu'on attend théoriquement. La bibliothèque livrée peut facilement être sujet à des développements futurs pour y ajouter d'autres algorithmes d'extraction de caractéristiques comme l'algorithme SIFT<sup>6</sup> ou bien SURF<sup>7</sup> ou d'autres encore. Les bases de données générées par les algorithmes de la bibliothèque peuvent être exploitées dans

---

6. scale-invariant feature transform

7. speeded up robust features



les domaines CBIR et machine learning et peuvent être utilisées comme des entrées de dimension réduite à la place de l'image originale.

## Références

- [1] J. Eakins and M. Graham, “Content-based image retrieval,” 1999.
- [2] I. K. Sethi, I. L. Coman, and D. Stan, “Mining association rules between low-level image features and high-level concepts,” in *Aerospace/Defense Sensing, Simulation, and Controls*, pp. 279–290, International Society for Optics and Photonics, 2001.
- [3] S. Fischer, G. Cristóbal, and R. Redondo, “Sparse overcomplete gabor wavelet representation based on local competitions,” *IEEE Transactions on Image Processing*, vol. 15, no. 2, pp. 265–272, 2006.
- [4] M. J. Swain and D. H. Ballard, “Color indexing,” *International journal of computer vision*, vol. 7, no. 1, pp. 11–32, 1991.
- [5] T. Dharani and I. L. Aroquiaraj, “A survey on content based image retrieval,” in *Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on*, pp. 485–490, IEEE, 2013.
- [6] F. Rajam and S. Valli, “A survey on content based image retrieval,” *Life science journal*, vol. 10, no. 2, pp. 2475–2487, 2013.
- [7] Y. Liu, D. Zhang, G. Lu, and W.-Y. Ma, “A survey of content-based image retrieval with high-level semantics,” *Pattern recognition*, vol. 40, no. 1, pp. 262–282, 2007.
- [8] M. A. Stricker and A. Dimai, “Color indexing with weak spatial constraints,” in *Electronic Imaging : Science & Technology*, pp. 29–40, International Society for Optics and Photonics, 1996.
- [9] J. Puzicha, J. M. Buhmann, Y. Rubner, and C. Tomasi, “Empirical evaluation of dissimilarity measures for color and texture,” in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, pp. 1165–1172, IEEE, 1999.
- [10] A. Chalechale and A. Mertins, “Angular radial partitioning for edge image description,” in *Proc. 7th International Symposium on DSP for Communication Systems (DSPCS03)*, pp. 479–484, 2003.
- [11] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.
- [12] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

- [13] M.-K. Hu, “Visual pattern recognition by moment invariants,” *IRE transactions on information theory*, vol. 8, no. 2, pp. 179–187, 1962.
- [14] M. Bober, “Mpeg-7 visual shape descriptors,” *IEEE Transactions on circuits and systems for video technology*, vol. 11, no. 6, pp. 716–719, 2001.
- [15] M. R. Teague, “Image analysis via the general theory of moments,” *JOSA*, vol. 70, no. 8, pp. 920–930, 1980.
- [16] Itseez, *The OpenCV Reference Manual*, 2.4.9.0 ed., April 2014.
- [17] T. G. et al., *GNU Multiple Precision Arithmetic Library 4.1.2*, December 2002.