UNIVERSITAS INDONESIA

UNIVERSITÉ
DE BRETAGNE SUD

DESKRIPSI DAN INDEKSASI CITRA BERDASARKAN INTERES POIN

TESIS

ICHSAN
1006804016

FAKULTAS TEKNIK
PROGRAM TEKNIK ELEKTRO
DEPOK
JULI 2012

UNIVERSITAS INDONESIA

UNIVERSITÉ
DE BRETAGNE SUD

DESKRIPSI DAN INDEKSASI CITRA BERDASARKAN INTERES POIN

TESIS

Diajukan sebagai salah satu syarat untuk memperoleh gelar Magister Teknik

ICHSAN
1006804016

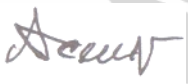
FAKULTAS TEKNIK
PROGRAM STUDI TEKNIK ELEKTRO
KEKHUSUSAN MULTIMEDIA DAN TEKNOLOGI INFORMASI
DEPOK
JULI 2012

HALAMAN PERNYATAAN ORISINALITAS

Tesis ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama : Ichsan

NPM : 1006804016

TandaTangan: 

Tanggal : 23 Juli 2012

HALAMAN PENGESAHAN

Tesis ini diajukan oleh :

Nama	:	Ichsan
NPM	:	1006804016
Program Studi	:	Teknik Elektro
Judul	:	Deskripsi dan Indeksasi Citra berdasarkan Interes Poin

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Magister Teknik pada Program Studi Teknik Elektro, Fakultas Teknik, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing : Prof. Sébastien Lefèvre

Penguji 1 : Dr. Luc Courtrai

Penguji 2 : Prof. Sylvie Gibet

Ditetapkan di : Vannes – Perancis

Tanggal : 29 Juni 2012

Mengetahui,

Ka. Departemen Teknik Elektro
Fakultas Teknik – Universitas Indonesia

Ir. Muhammad Asvial, MSc. PhD.

KATA PENGANTAR

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa, atas limpahan rahmat dah hidayah-nya, sehingga saya dapat menyelesaikan tesis ini. Penulisan tesis ini dilakukan dalam rangka memenuhi salah satu syarat untuk mencapai gelar ganda (*Double Degree*) Magister Teknik Program Studi Teknik Elektro pada Fakultas Teknik Universitas Indonesia dan Master 2 *Informatique de l'Image et des Réseaux (IIR)* pada UBS – Université de Bretagne-Sud, Perancis.

Saya menyadari bahwa, tanpa bantuan dan bimbingan dari berbagai pihak, dari masa perkuliahan sampai pada penyusunan tesis ini, sangatlah sulit bagi saya untuk menyelesaikan tesis ini. Oleh karena itu, saya mengucapkan terima kasih kepada:

1. Prof. Sébastien Lefèvre selaku dosen pembimbing yang telah menyediakan waktu, tenaga, dan pikiran untuk mengarahkan saya dalam penyusunan tesis ini.
2. Orang tua dan keluarga saya yang telah memberikan bantuan dukungan material dan moral.
3. Sahabat DDIP Vannes yang telah banyak membantu saya dalam menyelesaikan tesis ini.

Akhir kata, saya berharap Tuhan Yang Maha Esa berkenan membalas segala kebaikan semua pihak yang telah membantu. Semoga tesis ini membawa manfaat bagi pengembangan ilmu.

Depok, 23 Juli 2012

Penulis

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI
TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertandatangan di bawah ini:

Nama	: Ichsan
NPM	: 1006804016
Program Studi	: Teknik Jaringan Informasi dan Multimedia
Departemen	: Teknik Elektro
Fakultas	: Teknik
Jeniskarya	: Tesis

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Non eksklusif (*Non-exclusive Royalty Free Right*)** atas karya ilmiah saya yang berjudul : **Deskripsi dan Indeksasi Cintra berdasarkan Interes Poin** beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non eksklusif ini Universitas Indonesia berhak menyimpan, mengalih media/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 23 Juli 2012
Yang menyatakan

(Ichsan)

ABSTRAK

Nama : Ichsan
Program Studi : Teknik Elektro
Judul : Deskripsi dan Indeksasi Citra berdasarkan Interes Poin

Penelitian yang disajikan dalam laporan ini dilakukan di Laboratorium IRISA-UBS (Institut de Recherche en Informatique et Systèmes Aléatoires - Université de Bretagne Sud) dengan judul "Deskripsi dan Indeksasi Citra berdasarkan Interes Poin".

Tujuan dari proyek ini adalah untuk mempelajari metode dan sistem untuk membangun suatu sistem pengindeksasian dan pencarian citra berdasarkan fitur. Tujuan dari pengindeksasian citra adalah untuk menemukan citra yang sama dari citra dari database untuk suatu query citra. Setiap citra memiliki fitur, sehingga pengindeksasian citra dapat diimplementasikan dengan membandingkan fitur citra yang diambil dari gambar. Dalam proyek ini, akan dikembangkan aplikasi pencarian kemiripan citra berdasarkan gray-level histogram, aplikasi pencarian kemiripan citra yang menggunakan metode Sobel dan pengembangan sistem "*interest point*". Untuk mendeteksi "interest point" kita akan mengimplementasikan metode *Scale Invariant Feature Transform* (SIFT) pada lingkungan platform PELICAN.

Keywords :
Indeksasi citra, Kemiripan citra, Interest point, SIFT, PELICAN

ABSTRAITE

Nom et Prenom : Ichsan
Parcours : Genie Electrique
Titre : Description et Indexation d'Image par Points d'Intérêt

Les travaux de recherche présentés dans ce rapport ont été effectués au sein du laboratioire IRISA-UBS (Institut de Recherche en Informatique et Systèmes Aléatoires - Université de Bretagne Sud) sur le sujet << Description et indexation d'image par points d'intérêt >> .
L'objectif de mon travail est d'étudier l'état de l'art des méthodes et des systèmes pour construire un système l'indexation et de recherche d'image par le contenu. L'objectif de l'indexation des images est de récupérer des images similaires à partir d'une base de données image pour une image requête donnée. Chaque image a sa particularité. Ainsi l'indexation d'images peuvent être mis en oeuvre en comparant leur caractéristiques qui sont extraites à partir des images. Dans ce projet, je vais développer la similarite d'images par l'histogramme de niveaux de gris, la similarite d'image par l'histogramme qui utilise method Sobel et le système qui concerne sur le point d'intérêt. Pour détecter le point d'intérêt, nous appliquons la méthode Scale Invariant Feature Transform (SIFT) sous plate-forme PELICAN.

Mots-clés :
Indexation d'images, Image similaire, Point d'Intérêt , SIFT, PELICAN

ABSTRACT

Name : Ichsan

Study of Program : Electrical Engineering

Title : Description and Indexation Image based on Interest Point

The research presented in this report were carried out in Laboratoy IRISA-UBS (Institut de Recherche en Informatique et Systèmes Aléatoires - Université de Bretagne Sud) on the subject Description and Indexation Image based on Interest Point.

The purpose of my project is to study the method and system to build a system of indexing and searching an image based on the content. The objective of image indexation is to retrieve similar images from a database image to an image query. Each image has its feature. So, the indexing of image can be implemented by comparing their feature which extracted from the image. In this project, i will develop the similarity of images based on gray-level histogram, the similarity of images using method Sobel and the system concern the interest point. For detection the interest point, we implement the method Scale Invariant Feature Transform (SIFT) under platform PELICAN.

Keywords :

Image indexation, Similarity image, Interest point, SIFT, PELICAN.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

CHAPITRE 1
INTRODUCTION

1.1. Présentation du projet

Le travail à effectuer dans ce stage se situe dans le domaine de la recherche d'images. Aujourd'hui, le nombre d'images disponibles sur le World Wide Web est très grand, et il continue de plus en plus chaque jour. Les moteurs de recherche d'images sont considérés comme des outils très utiles qui permettent aux gens d'accéder facilement à cette information et en tirer profit. L'une des techniques de recherche d'images est l'approche concerne le contenu de l'image telle que la couleur et la texture. Cette approche, connue sous le nom en l'anglais de « Content-based Image Retrieval » ou CBIR, a été proposée au début des années 90 et vient de la communauté de vision par ordinateur.

Dans le développement ce système, nous mettons en oeuvre plate-forme PELICAN (Polyvalent Extensible Library for Image Computing and Analysis). Cette plate-forme est régulièrement utilisée comme un dépôt commun d'algorithmes de traitement d'image par des chercheurs et étudiants, dans un but de recherché ou d'enseignement. Avec cette plate-forme, nous pouvons faire des applications de traitement d'image en utilisant une bibliothèque qui a été défini dans le plate-forme Pélican. La mission de ce projet est la mise en œuvre de l'algorithme Scale Invariant Feature Transform (SIFT) sous la plate-forme Pelican.

Pour arriver l'objectif, la première étape consiste à apprendre le traitement des images. Dans l'étude de traitement d'image, on concentre recherche d'images basé sur le contenu, utilise le contenu visuel d'une image comme la couleur, forme, la texture pour représenter de l'indice de l'image. Le but est de trouver des caractéristiques permettant d'indexer l'image et de retrouver des images similaires lors d'une requête en fonction du contenu des images.

La deuxième étape dans ce projet consiste à se familiariser avec l'utilisation de la plate-forme PELICAN utilisé dans le projet principal. Dans cette

étape , nous allons étudier la plate-forme PELICAN est basée sur le paradigme orienté-objet et la technologie Java, et son architecture est modulaire et composée de trois couches : le noyau, les algorithmes, et les interfaces. La dernière partie du projet est la mis en œuvre la similarite d'images par l'histogramme, la similarite d'image par l'histogramme gradient et la méthode Scale Invariant Feature Transform (SIFT). La méthode Scale Invariant Feature Transform (SIFT) : Il s'agit d'informations numériques dérivées de l'analyse locale d'une image et qui caractérisent le contenu visuel de cette image de la façon la plus indépendante possible de l'échelle (« zoom » et résolution du capteur), du cadrage, de l'angle d'observation et de l'exposition (luminosité).

1.2. Les limites de notre travail

Pour clarifier les limites de notre travail, alors nous pourrions décrire comme ci-dessous :

1. Tous les algorithmes sont réalisés dans la plate-forme PELICAN
2. Développement d'une algortihme de la recherche d'images similaires basé sur l'histogramme de l'image en niveaux de gris.
3. Développement d'une algortihme de la recherche d'images similaires basé sur l'histogramme qui utilise méthod Sobel.
4. Développement d'une algortihme de détection de points d'intérêt qui utilise méthod SIFT.

1.3. Données de notre travail

Lors du test notre algorithme, nous avons besoin les données. Toutes les données donc nous utilison dans ce travai sont les données obtenus de Computer Vision Test Images.

CHAPITRE 2
CONTEXTE

Ce chapitre contient un profil du laboratoire où nous travaillons, présentation de l'équipe SEASIDE , présentation general de l'historique de PELICAN et l'architecture PELICAN.

2.1. IRISA - UBS

L'IRISA-UBS est un laboratoire de recherce en informatique de l'Université de Bretagne Sud, développe ses activités dans le domaine de l'informatique diffuse et de "l'intelligence ambiante" en intégrant trois voies complémentaires de recherche:

- Les systèmes logiciels interactifs multimédia et "intelligents" en tant que support à une "intelligence ambiante".
- L'architecture des systèmes logiciels dédiée à la maintenance, au test et à l'évolution dynamique des composants distribués en tant que support à une "informatique ambiante".
- Les intergiciels pour les systèmes distribués mobiles et communicants en tant que support à une "informatique ubiquitaire et diffuse".

L'IRISA-UBS est structuré en 3 équipes :

1. ARCHWARE (architectures logicielles).
2. CASA (intergiciels pour l'informatique mobile).
3. SEASIDE (fouille, analyse, synthèse de données complexes multimédia et interaction).

2.1.1. SEASIDE

Les chercheurs de l'équipe SEASIDE (SEarch, Analyse, Simulate and Interact with Data Ecosystems) étudient les moyens de [2] :

- Représenter et de modéliser les données multimédia complexes, caractérisées par différentes sources d'information hétérogènes, allant des données

numériques multidimensionnelles et potentiellement temporelles à des données sémantiques multi-niveaux.

- Fournir des mécanismes d'indexation et de recherche efficaces et intelligents pour ces données.
- Proposer des architectures distribuées adaptées.

2.2. Présentation général de PELICAN

2.2.1. Introduction

PELICAN est une plate-forme permettant la réalisation et l'exécution d'outils d'analyse et de traitement des images. Cette plate-forme est implémentée en Java. Elle est utilisée dans le cadre des travaux de recherche en traitements d'image et permet la conception, l'expérimentation et l'utilisation de méthodes de traitements d'images dans un environnement générique.

2.2.2. L'historique de PELICAN

Le Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection (LSIIT, UMR 7005 CNRS / ULP) Université de Strasbourg développe depuis 2004 un environnement pour la réalisation et l'exécution d'outils d'analyse et de traitement des images. Cet environnement a été intitulé PELICAN pour Polyvalent Extensible Library for Image Computing and ANalysis. PELICAN consiste actuellement en plusieurs centaines de classes Java permettant le traitement d'images de différentes natures (2-D, 3-D, images couleurs ou multispectrales, séquences vidéo, etc) dans des domaines d'applications varies (imagerie médicale, imagerie satellite, imagerie astronomique, recherche par le contenu, indexation vidéo, etc). Outre ces classes, des modules expérimentaux ont été élaborés pour l'exécution en parallèle, l'accès à distance, la programmation visuelle, etc.

Pour la réalisation de projet PELICAN , il y a deux différentes branches de développement au sein de la même équipe: La branche «recherche» qui a pour but de développer les algorithmes de traitement d'image (dans différents domaines tel que Télédétection, Morphologie, Segmentation...) et la branche «génie logiciel» qui développe la plate-forme PELICAN.

Côté réalisation des algorithmes de traitement d'images:

- Sébastien Lefèvre (Maitre de conferences, Chef de projet sur PELICAN)
- Benjamin Perret (Doctorant)
- Jonathan Weber (Doctorant)

Côté réalisation de la plate-forme PELICAN:

- Vincent Danner (Stagiaire)

Anciens membres :

- Erchan Aptoula (Doctorant)
- Sébastien Derivaux (Doctorant)
- Florent Sollier (Apprenti)
- Régis Witz (Ingénieur)

Les activités de recherche de l'équipe sont centrées sur l'imagerie, principalement le traitement et l'analyse, mais également sur les interactions de celles-ci avec la synthèse. Elles s'appuient sur de solides bases mathématiques et visent particulièrement les développements algorithmiques novateurs. L'unité de l'équipe porte sur les aspects méthodologiques de l'analyse multi-images au sens large, autour de l'acquisition, du traitement, de l'interprétation et de l'exploitation de séquences d'images (temporelles, spatiales, multimodales, multispectrales).

Les applications portent sur l'analyse de séquences vidéo, l'imagerie médicale, l'imagerie astronomique, la réalité augmentée ou enrichie, la vision industrielle et la métrologie.

2.2.3. L'architecture PELICAN

L'architecture de PELICAN est modulaire et composée de trois couches (figure 2.1) : le noyau, les algorithmes, et les interfaces :

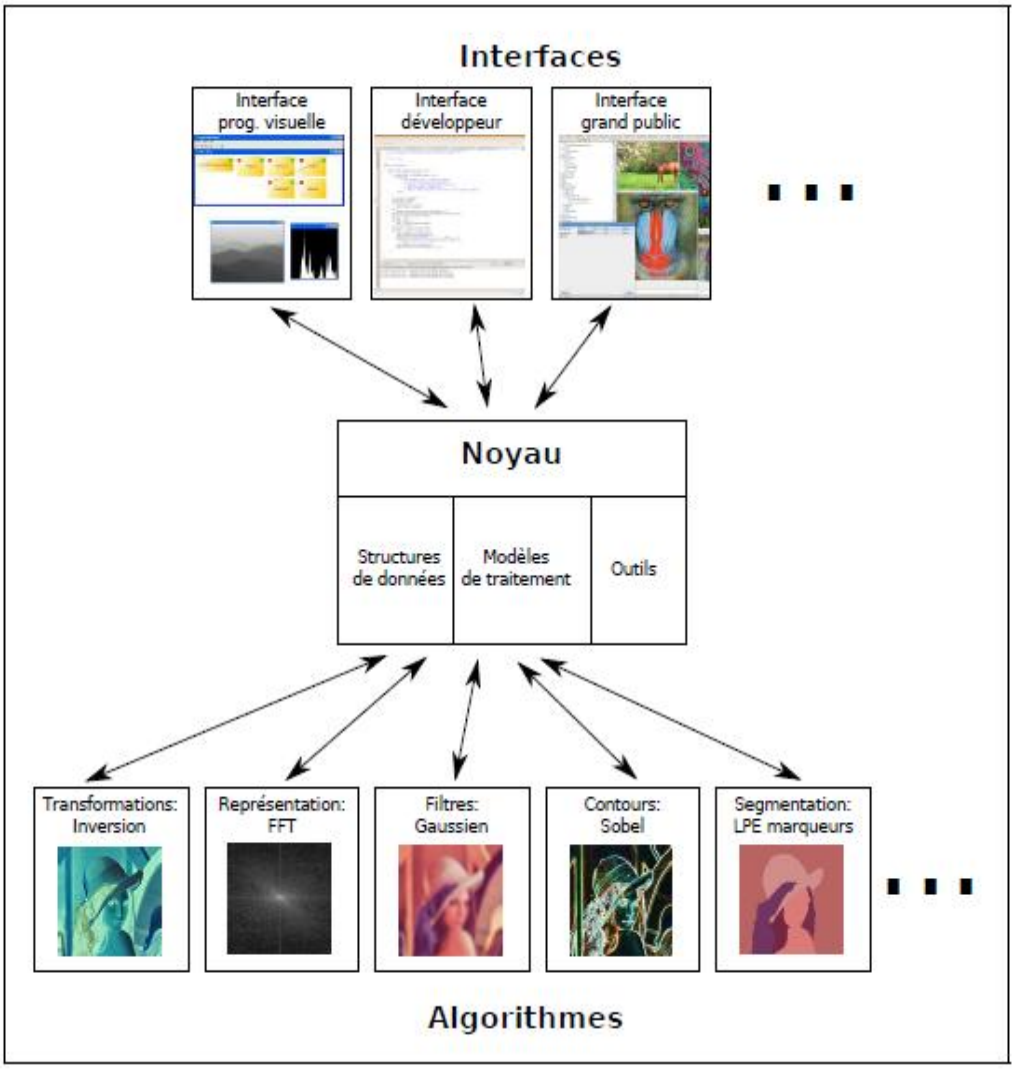


Figure 2.1- l'architecture PELICAN

1. Le noyau

   Le noyau est la partie centrale de la plate-forme, il contient l'ensemble des structures de données et des fonctions pouvant être utilisées par les éléments des autres couches. Médiateur situé entre les algorithmes et les interfaces, il évite une communication directe entre ces deux parties : il assure ainsi qu'aucune modification effectuée sur un algorithme ou une interface n'aura une répercussion sur la partie opposée, et permet de rendre les interfaces complètement indépendantes des algorithmes.

2. Les algorithmes

   Les algorithmes sont les techniques d'analyse et de traitement d'images présentes dans la plate-forme sous forme de classes : il peut s'agir de

contributions originales de chercheurs de l'équipe ou d'implantations d'outils publiés dans la littérature. L'écriture d'un algorithme est facilitée par les fonctionnalités offertes par le noyau, mais doit suivre un ensemble de règles prédéfinies, ce qui permet d'assurer sa compatibilité avec les autres éléments de la plate-forme.

3. Les interfaces

   Les interfaces, dont une liste non-exhaustive sera donnée dans la prochaine section, permettent l'accès à la plateforme et à ses algorithmes. Elles répondent à des cas d'utilisation spécifiques et sont adaptées aux différents profils d'utilisateurs.

PELICAN est décomposé en quatre sous-doussiers qui sont les suivants :

- Algorithmes

  Ce dossier correspond à l'ensemble des algorithmes de traitement d'image qui ont étés par les doctorants de l'équipe MIV. Pour assures une recherche rapide d'un algorithme particulier et garantir une certaine hiérachie, le dossier « algorithm » est composé de plusieurs sous-doussiers (« arithmetic », « detection », « geometric », « segmentation », etc..) qui décrivent la nature des algorithmes contenus.
- Gui

  Ce dossier regroupe l'ensemble des Interfaces Homme-Machine (IHM) disponibles ainsi que des modules de visualation d'image.
- Demos

  Ce dossier contient un ensemble de classes de test qui montrent le bon fonctionnement d'algorithmes spécifiques ou d'applications plus complexes.
- Pelican

  Pelican est le dossier principal qui inclut tous les dossiers décrit ci-dessus, ce dossier contient également toutes les classes de base du project : Les classes d'image (Image, BooleanImage, ByteImage, IntegerImage et DoubleImage), la classe abstraite Algorithm qui est la classe mère de tous les algorithmes de traitement d'image, et enfin, les classes d'erreurs.

Une image est représentée par la classe Image qui dispose d'un grand nombre d'accesseurs. On peut accéder au contenu de l'image selon plusieurs types de données et plusieurs types d'accès.

Types de données:

- bool

  Vrai ou faux, le blanc correspond à vrai et le noir à faux (image binaire).
- byte

  Codé sur un entier avec une valeur appartenant à [0;255] (différents de Byte.MIN_VALUE et Byte.MAX_VALUE qui sont signés) .
- int

  Codé sur un entire avec une valeur appartenant à [Integer.MIN_VALUE; Integer.MAX_VALUE].
- double

  Codé sur un double avec une valeur comprise dans [0.0; 1.0].

Par exemple, si on accède au même pixel d'intensité maximale avec les différents type d'accès :

- getPixelBoolean( ) retourne vrai.
- getPixelByte( ) retourne 255.
- getPixelInt( ) retourne Integer.MAX_VALUE.
- getPixelDouble( ) retourne 1.0.

La classe Image est abstraite car plusieurs types de stockage effectif sont disponibles. Parmi ceux-ci on a les classes BooleanImage, ByteImage, IntegerImage, DoubleImage qui stockent chacune les pixels au format bool, byte, int et double. Les formats de stockage n'ont rien à voir avec les formats d'accès même si on retrouve les types de bases. Une BooleanImage pourra être accédé avec getPixelDouble( ) et inversement une DoubleImage pourra être accédée avec getPixelBoolean( ). Plus spécifiquement une ByteImage stocke ses pixels sur des bytes donc dans un interval entre Byte.MIN_VALUE(128) et Byte.MAX_VALUE (+127) alors que getPixelByte( ) retourne [0;255].

La classe Image fournit une implémentation générale et un comportement standard aux classes filles mais ne possède pas les données des pixels qui sont incluses dans les classes filles. Ces sousclasses possèdent leurs méthodes et propriétés spécifiques. Voici quelques exemples qui illustrent les utilisations de ces différents types de stockage:

- ByteImage
  Recherche par le contenu, simplication, segmentation, etc..
- IntegerImage
  Imagerie médicale, IRM.
- DoubleImage
  L'imagerie astronomique est bon exemple d'utilisation de ce format car les telescopes peuvent renvoyer tout type de valeur.
- BooleanImage
  Ce format existe dans le système seulement pour faciliter les algorithmes traitant des images binaires, les seuillages par exemple

2.2.4 Comment utilser Pelican dans IDE Eclipse

- Installer Maven d'abord sur le système. Le Maven peut être téléchargé depuis http://maven.apache.org/download.html.
- Suivre les instructions d'installation ou « sudo apt-get install maven2 » pour Linux.
- Installer le plug-in IAM depuis http://q4e.googlecode.com/svn/trunk/updatesite-iam via le mode d'installation de plug-in d'eclipse

CHAPITRE 3
LES BASES THÉORIQUES

3.1. Les règles pour créer un algorithme dans PELICAN

1. Tous les attributs qui sont des paramètres en entrée (obligatoire ou optionnel) et en sortie de l'algorithme doivent être à visibilité publique car ils doivent être accessibles depuis la classe abstraite Algorithm. Tous les autres attributs locaux peuvent rester avec une visibilité privée.

2. Il y a seulement trois méthodes publiques :
   a. Le constructeur

      Dans le constructeur on renseigne les différents attributs de type String de la classe Algorithm: super.inputs qui contient tous les noms des attributs d'entrée obligatoires séparés par une virgule sans espace ; super.outputs qui contient tous les noms des attributs de sortie séparés par une virgule sans espace ; et enfin super.options qui contient tous les noms des attributs d'entrée optionnels séparés par une virgule sans espace. Dans tous les cas l'ordre dans lequel les noms des attributs sont fournis est très important, puisque l'ordre des noms dans super.inputs (et de façon optionnelle dans super.options) doit correspondre à l'ordre dans lesquels les paramètres seront fournis lors d'un appel à process(), l'ordre des noms dans super.outputs doit correspondre à l'ordre dans lesquels les objets de sortie seront retournés lors d'un appel à process, processAll et processOne (voir la classe Algorithm pour plus de détails).

   b. Le launch

      La méthode launch contient le code de l'algorithme et doit vérifier la validité du contenu des paramètres (seuls les types sont vérifiés automatiquement par la classe Algorithm).

   c. Le exec (optionnel)

      Pour avoir une méthode utilitaire de classe (statique) fortement typée, on ajoute une méthode intitulée exec dans la classe concrète pour faire

un appel à la méthode process définie dans la classe abstraite ; par exemple :

```
public Image exec(Image im, int val)
{
    return (Image) new Exemple().process(im,val);
}
```

Cette classe doit prendre en paramètre tous les attributs d'entrée en gardant leurs noms et leur ordre définis dans super.inputs dans le constructeur

3. La Javadoc doit être renseignée (au minimum) pour
   a. La classe

      Un commentaire de quelques lignes qui décrit quel traitement fait l'algorithme sur quel type d'objet et ce qu'il renvoie. Le champ @author doit être renseigné, il comprend en premier lieu le nom du créateur de la classe et, s'il y en a, les nom des personnes qui ont apporté des modifications (séparés par une virgule).
   b. Chaque attribut d'entrée, de sortie et d'option
   c. La méthode exec:

      Cette Javadoc comprend la description du traitement que fait cette méthode (copier/coller de la Javadoc de la classe ?), une balise @param doit être définie pour chaque paramètre de cette méthode, une balise @return doit être définie pour l'objet de retour.

Dans le PELICAN tous les algorithmes implémentent la classe Algorithm, c'est donc à partir de cette classe qu'on a pu mettre en place l'introspection. Le but était de pouvoir utiliser des méthodes spécifiques à chaque algorithme sans avoir à les écrire dans chaque classe. L'usage de l'introspection nous a permis de pouvoir récupérer les types, les noms et les valeurs des entrées, sorties et des options. Elle nous permet également de lancer le traitement de l'algorithme. Ainsi chaque

algorithme, quelque soit ses spécificités est traité de la même manière que les autres, on obtiens alors une architecture générique.

```
package fr.unistra.pelican.algorithms.arithmetic;

import fr.unistra.pelican.Algorithm;
import fr.unistra.pelican.AlgorithmException;
import fr.unistra.pelican.Image;
import fr.unistra.pelican.InvalidParameterException;

/**
 * Compute the addition beetwen two images, inputImage1 + inputImage2. The
 * outputImage format is the same as inputImage1. No check is done on value.
 *
 * @author ?, Benjamin Perret
 */
public class Addition extends Algorithm {

    /**
     * First input image.
     */
    public Image inputImage1;

    /**
     * Second input image.
     */
    public Image inputImage2;

    /**
     * Algorithm result: addition of image one and two.
     */
    public Image outputImage;

    /**
     * Constructor
     *
     */
    public Addition() {
       super();
       super.inputs = "inputImage1,inputImage2";
       super.outputs = "outputImage";
    }

    /**
     * Compute the addition beetwen two images, inputImage1 + inputImage2. The
     * outputImage format is the same as inputImage1.
     */
    public void launch() throws AlgorithmException {

    outputImage = inputImage1.copyImage(false);
    int size = inputImage1.size();

    for (int i = 0; i < size; ++i)
       outputImage.setPixelDouble(i, inputImage1.getPixelDouble(i)
       + inputImage2.getPixelDouble(i));
```

```
    }


    /**
     * This algorithm Compute the addition between two images (inputImage1 +
     * inputImage2) which is the outputImage.
     *
     * @param inputImage1
     *            First of the two additioned images.
     * @param inputImage2
     *            Second of the two additioned images.
     * @return outputImage which is the addition between the two images.
     */
    public static Image exec(Image inputImage1, Image inputImage2) {
    return (Image) new Addition().process(inputImage1, inputImage2);
    }
}
```

Figure.3.1 - l'exemple du code d'un algorithme

La figure.3.1 illustre toutes de l'exemple du code d'un algorithme simple . On a au début de la classe la déclaration des attributs: inputImage1 et inputImage2 qui sont les deux attributs d'entrée et outputImage qui est l'attribut de sortie. Vient ensuite le constructeur (public Addition()) où sont renseignés les différents attributs de type String avec les noms des attributs d'entrée et de sortie de l'algorithme Addition.

Et pour terminer la classe on a la méthode launch() (public void launch()) où est implémenté le traitement de l'algorithme Addition. C'est donc ici que ce fera l'addition de tous les pixels des images d'entrée pour mettre le résultat dans l'image de sortie.

La standardisation a été mise en place à travers un certain nombre de règles, ces règles vont ici être brièvement décrites:

1. Tous les attributs qui sont des paramètres d'entrée (obligatoire ou optionnel) et de sortie de l'algorithme doivent être à visibilité publique car ils doivent être accessibles depuis la classe abstraite Algorithm (par introspection).
2. Dans le constructeur il faut renseigner les différents attributs de type String de la classe Algorithm:
   - super.inputs qui contient tous les noms des attributs d'entrée obligatoires séparés par une virgule sans espace.
   - super.outputs qui contient tous les noms des attributs de sortie séparés par une virgule sans espace.

– super.options qui contient tous les noms des attributs d'entrée optionnels séparés par une virgule sans espace

Grâce à cette spécification on va récupérer le nom des entrées/options/sorties par introspection et avec noms on va récupérer, toujours par introspection, la valeur de ces mêmes entrées/options/sorties qui sont en visibilité publique.

3. La Javadoc doit être renseignée (au minimum) pour la classe et chaque attribut d'entrée, de sortie et d'option.

   Il faut toujours que la Javadoc soit renseignée car elle est récupérée automatiquement par expression régulière et réinjectée dans les interfaces graphiques pour donner un maximum de compréhension des algorithmes à l'utilisateur. Le résultat de cette standardisation est une grande souplesse pour la personne qui va créer un algorithme, puisque pour que son algorithme fonctionne avec n'importe quelle IHM il lui suffira de renseigner ce peu d'information et ensuite son code sera compatible.

3.2. L'histogramme d'une image

Un histogramme est en fait un graphique avec des colonnes; de nombreux types de données peuvent être représentées dans ce type de diagramme. Quand on parle d'images numériques, un histogramme est un diagramme représentant la distribution des tons des pixels dans une image donnée. Comme nous pouvont représenté dans un diagramme la taille des personnes dans une classe nous pouvons représenter la "luminositée des pixels dans une image. L'ordinateur compte tous les pixels dans une image qui ont une luminosité particulière et les représentent dans le diagramme.

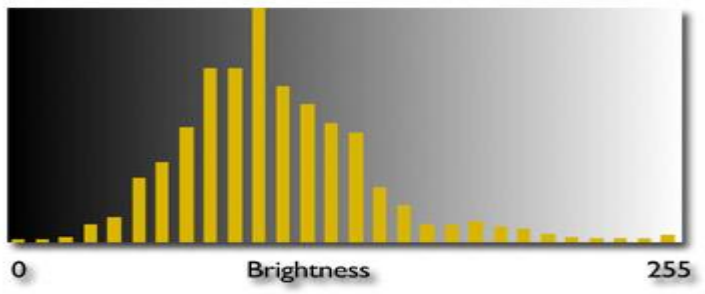


Figure 3.2 – L'histogramme d'image

La luminosité des pixels (sur une échelle standard allant de 0 à 255) est tracé sur l'axe horizontal "X". L'axe vertical "Y" représente le nombre de pixel correspondant un ton particulier. Plus la barre est haute plus il y a de pixels dans ce ton particulier. Lire l'histogramme ci-dessus montre qu'il y a beaucoup de pixels dans la gamme des tons gris moyen et qu'il y a très peu dans la gamme plus lumineuse et sombre. Généralement un histogramme est tracé avec un ensemble de barres verticales qui se touchent (car il y a beaucoup de points à tracer). L'exemple ci-dessus est un exemple simple à lire parce que les barres verticales sont espacées. Un histogramme d'image est un outil très utile pour juger l'exposition d'une photo.

Remarquez comment il y a un pic de données à la cauche du centre de l'histogramme. Cela représente tous les pixels noirs qui entourent la jeune femme. Comme il y a peu de pixels très lumineux il y a peu de données sur le côté droit du graphe

3.3. CBIR

La recherche d'images par le contenu (en anglais : Content Based Image Retrieval ou CBIR), est une technique permettant de rechercher des images à partir de ses caractéristiques visuelles, c'est-à-dire induite de leurs pixels. Les images sont classiquement décrites comme rendant compte de leur texture, couleur, forme [1].

Un cas typique d'utilisation est la recherche par l'exemple où l'on souhaite retrouver des images visuellement similaire à un exemple donné en requête. Il s'oppose à la recherche d'images par mots clés ou tags, qui fut historiquement proposé par les moteurs de recherche tels que google image grâce à des banques d'images où les images sont retrouvées en utilisant le texte qui les accompagne plutôt que le contenu de l'image elle-même (google image propose désormais des filtres basés sur le contenu pixelique des images).

Le principe général de la recherche d'image par le contenu se déroule en deux temps. Lors d'une première phase hors ligne (étape d'indexation), on calcule les signatures des images et on les stocke dans une base de donnée. La seconde phase, dite de recherche se déroule en ligne. L'utilisateur soumet une image comme requête. Le système calcule la signature selon le même mode que lors de la

première phase d'indexation. Puis, cette signature est comparée à l'ensemble des signatures préalablement stockées pour en ramener les images les plus semblables à la requête. Lors de la phase d'indexation, le calcul de signature consiste en l'extraction de caractéristiques visuelles des images telles que :

- la texture (filtre de Gabor, transformée en ondelettes discrète…)
- la couleur (histogramme de couleurs, histogrammes dans l'espace RGB, TSV).
- les formes (descripteurs de Fourier).
- une combinaison de plusieurs de ces caractéristiques.

Ces caractéristiques sont dites de bas-niveau, car elles sont très proches du signal, et ne véhiculent pas de sémantique particulière sur l'image.

Une fois ces caractéristiques extraites, la comparaison consiste généralement à définir diverses distances entre ces caractéristiques, et de définir une mesure de similarité globales entre deux images. Au moyen de cette mesure de similarité et d'une image requête, on peut alors calculer l'ensemble des mesures de similarités entre cette image requête et l'ensemble des images de la base d'images. On peut alors ordonner les images de la base suivant leur score, et présenter le résultat à l'utilisateur, les images de plus grand score étant considérées comme les plus similaires.

Ce genre de système permet aussi de rechercher des images sans forcément avoir une image requête, par exemple rechercher des images plutôt bleues, ou alors dessiner une forme et demander de chercher toutes les images qui possèdent un objet de forme similaire

3.4. Points d'intérêt

La détection de points d'intérêt dans les images est de plus en plus utilisé afin de facilité de nombreuses tâches : reconnaissance d'objet, assemblage d'images, modélisation 3D, la recherche d'image par le contenu, le tracking video, etc. Les points clés extraits d'une image permettent de caractériser cette image. En comparant les points clés d'une image et ceux d'une autre image, on peut alors déduire si des informations communes sont présentes dans ces deux images.

Le descripteur SIFT est le plus couramment utilisé pour l'extraction de points clés. Ce qui a fait le succès de ce descripteur, c'est qu'il est peu sensible au changement d'intensité, de mise à l'échelle et de rotation, ce qui fait de lui un descripteur très robuste. Il est basé sur les différences de gaussiennes [9].

3.5. La méthode SIFT

Scale-invariant feature transform (SIFT), que l'on peut traduire par « transformation de caractéristiques visuelles invariante à l'échelle », est un algorithme utilisé dans le domaine de la vision par ordinateur pour détecter et identifier les éléments similaires entre différentes images numériques (éléments de paysages, objets, personnes, etc.). Il a été développé en 1999 par le chercheur David Lowe. L'étape fondamentale de la méthode proposée par Lowe consiste à calculer ce que l'on appelle les « descripteurs SIFT » des images à étudier. Il s'agit d'informations numériques dérivées de l'analyse locale d'une image et qui caractérisent le contenu visuel de cette image de la façon la plus indépendante possible de l'échelle (« zoom » et résolution du capteur), du cadrage, de l'angle d'observation et de l'exposition (luminosité).

Par exemple :

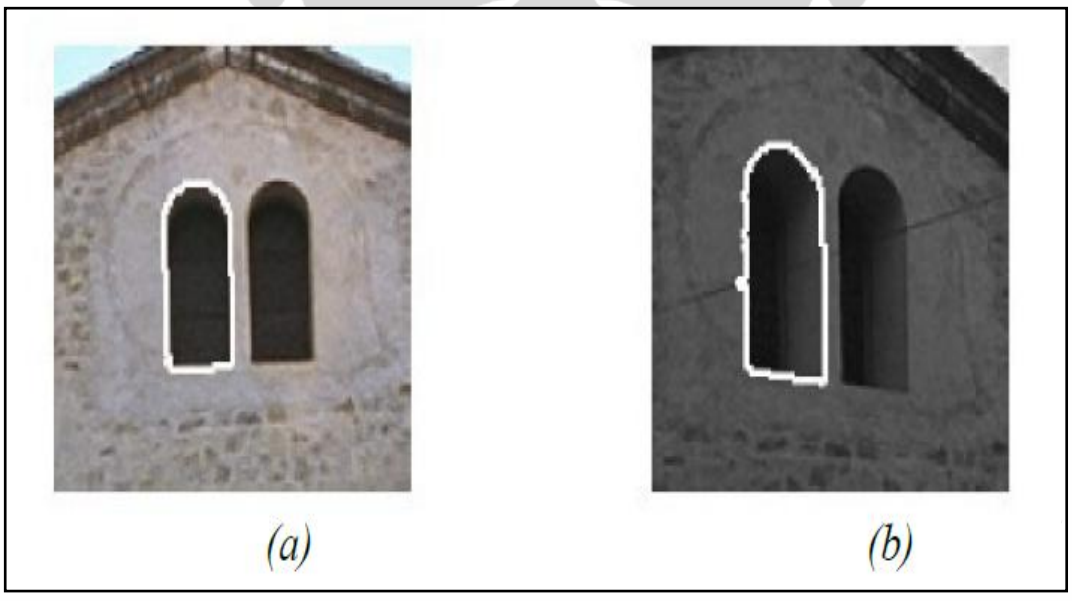


Image (a) et image (b) représentent une même scène mais en échelle differéntes en rotations differéntes et en lumière differéntes. La région qui représente la fenêtre à gauche dans image (a) est une région invariante car elle est aussi détecté dans l'image (b). De même, la région représentant la fenêtre à gauche dans image (b) est une région invariante.

L'approche SIFT, pour la generations de caractéristique l'image, nous prenons une image et la transforme en un «vaste collection de vecteurs de caractéristiques locales». Chacun de ces vecteurs de caractéristiques est invariant à toute l'échelle, la rotation ou la translation de l'image. Pour extraire de ces caractéristiques, l'algorithme SIFT applique 4 étapes de filtrage :

1. Détection de l'extrema de l'espace échelle :

   Cette étape consiste en fait à identifier l'échelle qui présente une invariance par rapport au redimensionnement de l'image. David Lowe a suggéré la recherche de l'extremum local dans l'espace échelle des différences de gaussiennes. Ainsi les caractéristiques SIFT sont localisées dans l'extrémum de l'espace échelle de la fonction de différences de gaussiennes. L'espace échelle d'une image est défini par une fonction $L = (x, y, \sigma)$ produite grâce à la convolution entre la gaussienne d'échelle variable $G = (x, y, \sigma)$ avec l'image d'entrée $I(x, y)$ [8].

   $$L = (x, y, \sigma) = G(x, y, \sigma) * (x, y)$$

   Avec

   $$G = (x, y, \sigma = \frac{1}{2\pi\sigma^2})e^{-(x^2+y^2)/2\sigma^2}$$

   Ainsi pour pouvoir extraire des points clés stables dans l'espace échelle, Lowe a proposé d'utiliser l'extrémum de la fonction de différences de gaussiennes convoluée à l'image $D(x, y, \sigma)$ qui peut être calculée à partir de la différence de deux échelles voisines séparées d'une constante multiplicative k :

   $$D = (x, y, \sigma) = (G = (x, y, k\sigma) - G = (x, y, \sigma))) * I(x, y)$$

   Ce qui donne :

$$D = (x, y, \sigma) = (L = (x, y, k\sigma) - L = (x, y, \sigma)))$$

La figure suivante ilustre l'approche de calcul de la DoG:

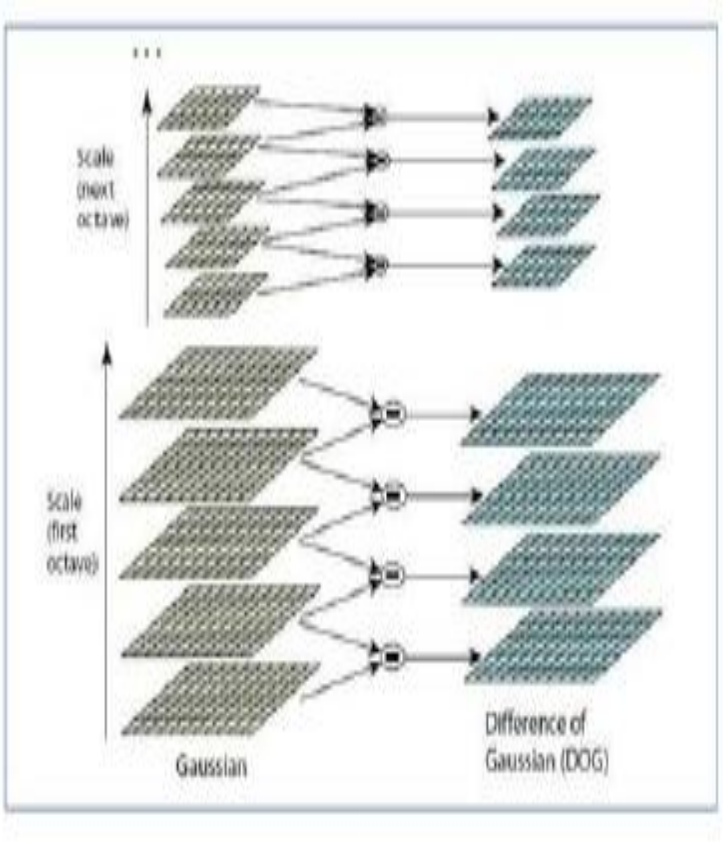


Figure.3.3 – Calcul des images de differences de gaussiennes

2. Détection de l'extrema local

   Dans le but de déterminer le maximum et le minimum local, chaque point d'échantillonnage est comparé à ses huit voisins dans l'image en question et aux neufs homologues dans l'échelle en dessus et en dessous. Ce point n'est alors choisi que lorsqu'il est plus grand que tous ses voisins ou plus petit que eux tous.

   La figure suivante ilustre l'approche de déterminer le maximum et le minimum local:

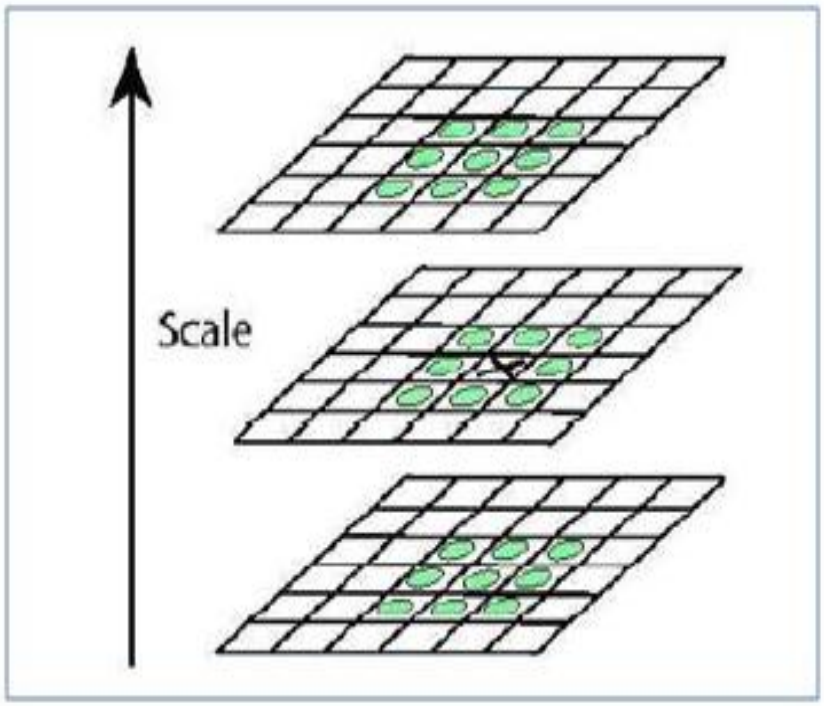


Figure.3.4 - Détermination de l'éxtrémum local

3. Localisation des points clés

   Les points clés qui sont générees à l'étape précédente produit un grand nombre de points clés. Certains d'entre eux ,ils ne sont pas stable. L'étape suivante consiste à éliminer les point clés qui ont une faible contraste. Pour éliminer les points-clés qui ont des valeurs de contraste faibles, l'expansion de Taylor du second ordre de D (x) calculée à offset $\hat{\mathbf{x}}$. Si la valeur est inférieure 0.03, puis le candidat de point clé est mis au rebut. En outre, la différence de gaussiennes présenterait une forte réponse tout au long des contours, même si cette localisation est trés instable vis-à-vis du bruit. Ainsi une image pauvrement défini aura une large et principale courbure vers le contour mais une petite dans la direction perpendiculaire. Les courbures principales peuvent être calculées à partir d'une matrice 2*2 de Hessian dans la position et l'échelle du point clé correspondant

$$\begin{pmatrix} D_{xx} D_{xy} \\ D_{xY} D_{yy} \end{pmatrix}$$

Les points clés présentant un ratio de courbures principal inférieur à un certain seuil sont rejetés.

4. Descripteur de point-clé

Le descripteur SIFT affecte à chaque caractéristique une localisation dans l'image, une échelle d'une orientation et un vecteur descripteur basés sur les propriétés locales de l'image. Le vecteur descripteur, étant donné qu'il est représenté relativement à l' échelle et à l'orientation, est invariant par rapport aux changement de ces deux paramètres. L'échelle du point clé est utilisée pour sélectionner l'image gaussienne lissée avec l' échelle la plus proche de façon à effectuer les calculs de manière invariante par rapport au redimensionnement. Pour chaque échantillon d'image $L(x, y)$ à cette échelle $m(x, y)$ ainsi que l'orientation $\theta(x, y)$ sont calculés en utilisant les différence entre les pixels.

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1}\left(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}\right)$$

Ainsi, il est possible de former des histogrammes d'orientation à partir des gradients d'orientation des points échantillonnés dans la région entourant le point clé. Un histogramme d'orientation contient 36 graduations correspondant aux 360 dégrés. Chaque échantillon ajouté à cet histogramme est pondéré par son gradient ainsi que par une fenêtre circulaire pondérée en gaussienne d'échelle 1.5 fois l' échelle du point. Par la suite, nous déterminos le plus pic de l'histogramme. En utilisant ce pic ainsi que le second pic présentant 80% du premier, nous créons un point clé avec son orientation. Finalement ces échantillons sont accumulés dans ces histogrammes d'orientation résumant le contenu sur 4 x 4 sous régions.

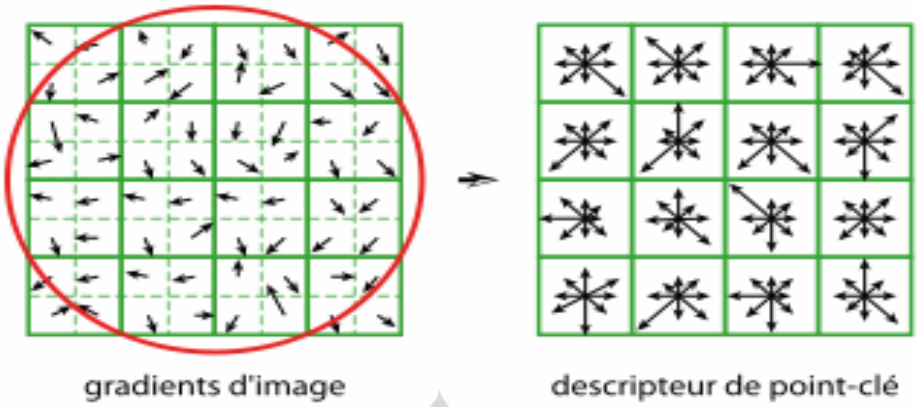


Figure.3.5-Construction d'un descripteur SIFT

Les SIFT caractéristiques sont obtenues par des matrices 4 x 4 d'histogrammes chacun comprenant 8 orientations différentes. Ceci qui fait au total un vecteur descripteur de 128 élements.

CHAPITRE 4
MES RÉALISATION

4.1. La Similarité d'Image

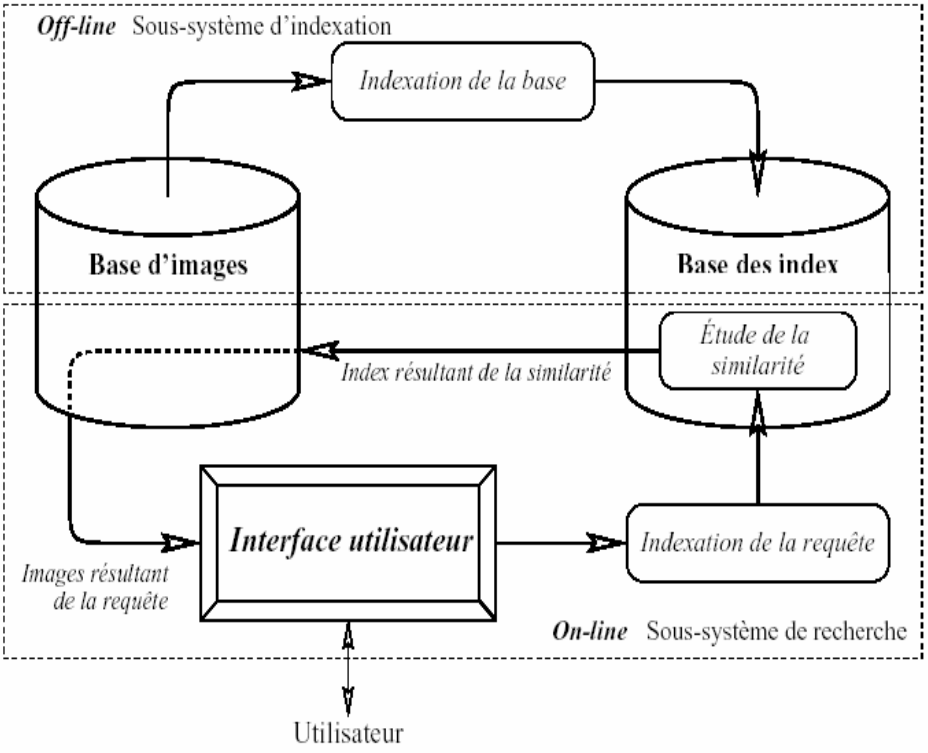


Figure.4.1 - L'architecture d'un système d'indexation et recherché d'images par le contenu

Ce système s'exécute en deux étapes : l'étape d'indexation et l'étape de recherche. Dans l'étape d'indexation, des caractéristiques sont automatiquement extraites à partir de l'image et stockées dans un vecteur numérique appelé descripteur visuel. Grâce aux techniques de la base de données, on peut stocker ces caractéristiques et les récupérer rapidement et efficacement.

Dans l'étape de recherche, le système prend une ou des requêtes à l'utilisateur et lui donne le résultat correspond à une liste d'images ordonnées en fonction de la

similarité entre leur descripteur visuel et celui de l'image requête en utilisant une mesure de distance.

4.1.1. L'histogramme des niveaux de gris

Avant nous implémentons la méthode Scale-invariant feature transform (SIFT) sous plate-forme PELICAN, tout d'abord on met en oeuvre le système de recherché d'images par similarité sur l'histogramme d'une image en niveaux de gris. L'objectif de cet expérience pour se familiariser avec la bibliothèque existante dans PELICAN et comment créer une classe Algorithm dans le plate-forme PELICAN. Dans cette expérience, nous implémentons la similarité par histogram d'une image en niveaux de gris. L'histogramme d'une image en 256 niveaux de gris sera représenté par un graphique possédant 256 valeurs en abscisses, et le nombre de pixels de la classe (niveau d'intensité) de l'image en ordonnées.

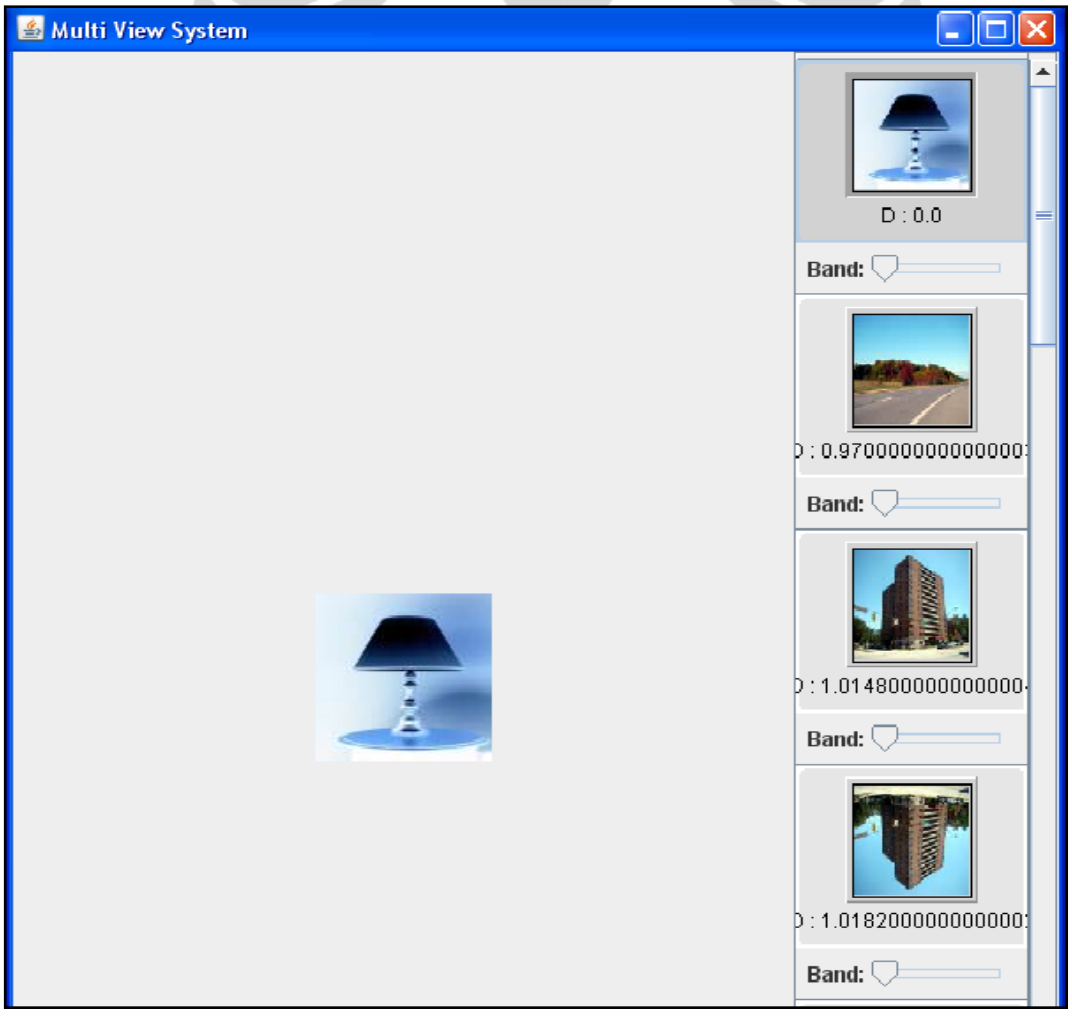


Figure.4.2 - l'application de la similarité d'image basé sur l'image niveau de gris

Dans cette application, nous convertissons d'abord l'image en mode "niveaux de gris" avec la bibliothèque plate-forme PELICAN soit : RGBToGray. Ensuite nous calculons l'histogramme de l'image.

```
public void launch() throws AlgorithmException {
        int size = 256;
        //double[] output = new double[size];
        output = new double[size];

        Image inputtmp = RGBToGray.exec(input);
        double[] anda = Histogram.exec(inputtmp);
        // length of histogram
        double[] histoOriginal = new double[anda.length];

        int i, j;
        // initialize all value of histogram to 0
        Arrays.fill(histoOriginal,0);
        for (i = 0; i < anda.length; i++) {
            histoOriginal[i] = 0;
        }

        // The "A" is array image (widht, height)
        int[][] A = new
int[inputtmp.getXDim()][inputtmp.getYDim()];

        // Compute the frequency the level of gray (0...255)
        for (int k = 0; k < 256; k++) {
            for (i = 0; i < inputtmp.getXDim(); i++) {
                for (j = 0; j < inputtmp.getYDim(); j++) {

                    A[i][j] = inputtmp.getPixelXYByte(i, j);
                        if (A[i][j] == k) {
                        histoOriginal[k] += 1;
                    }
                }
            }
        }

        for (i = 0; i < 256; i++) {
            output[i] = (histoOriginal[i] / inputtmp.size());
        }
    }
```

Figure.4.3 - Pseudo-code de calcul l'histogramme

Le valeur de cet histograme est utilisé pour comparer l'histogramme d'une image avec les autres histogramme de l'autres images par calcul la distance qui utilise la formule comme suit :

$$d(A,B) = \sum_{j=1}^{n} | H_j{}^A - H_j{}^B |$$

Nous calculons les distances entre l'histogramme de l'image requête et les histogrammes des autres images, le similarité quand trouve les images plus proches.

```
// Calculation the distance based on Histogram
        for (int i = 0; i < numotherfile; i++) {
            path[i] = others[i].getPath();
            nom[i] = others[i].getName();
            imothers[i] = ImageLoader.exec(path[i]);
            tab2 = IsanAlgo1.exec(imothers[i]);
            for (int d = 0; d < 256; d++) {
                double dis = Math.abs((tab[d]) - (tab2[d]));
                dis = Math.abs(dis);
                distance[i] = distance[i] + dis;
                evadistance[i] = evadistance[i] + dis;
            }
        }
        MultiView mv = new MultiView();
        mv = MViewer.exec();
        // sorting distance
        Arrays.sort(distance);
        System.out.println(Arrays.toString(distance));

        // Matching Image based on the closest distance
        for (int i = 0; i < numotherfile; i++) {
            for (int R = 0; R < numotherfile; R++) {
                if (distance[i] == evadistance[R]) {
                    Image im = ImageLoader.exec(path[R]);
                    im.setName("D : " + evadistance[R]);
                    mv.addImage(im);
                }
            }
        }
```

Figure.4.4 - Pseudo-code du processus de comparison l'histogramme

4.1.2. Sobel

Le filtre de Sobel est un opérateur utilisé en traitement d'image pour la détection de contours. Il s'agit d'un des opérateurs les plus simples qui donne toutefois des résultats corrects. Pour faire simple, l'opérateur calcule le gradient de l'intensité de chaque pixel. Ceci indique la direction de la plus forte variation du clair au sombre, ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords, ainsi que l'orientation de ces bords.

En termes mathématiques, le gradient d'une fonction de deux variables (ici l'intensité en fonction des coordonnées de l'image) est un vecteur de dimension 2 dont les coordonnées sont les dérivées selon les directions horizontale et verticale. En chaque point, le gradient pointe dans la direction du plus fort changement d'intensité, et sa longueur représente le taux de variation dans cette direction. Le gradient dans une zone d'intensité constante est donc nul. Au niveau d'un contour, le gradient traverse le contour, des intensités les plus sombres aux intensités les plus claires.

L'opérateur utilise des matrices de convolution. La matrice (généralement de taille 3×3) subit une convolution avec l'image pour calculer des approximations des dérivées horizontale et verticale. Soit A l'image source, G_x et Gy deux images qui en chaque point contiennent des approximations respectivement de la dérivée horizontale et verticale de chaque point. Ces images sont calculées comme suit:

$$Gx = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * A \text{ et } Gy = \begin{bmatrix} +1 & 2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

En chaque point, les approximations des gradients horizontaux et verticaux peuvent être combinées comme suit pour obtenir une approximation de la norme du gradient:

$$G = \sqrt{G_X{}^2 + G_Y{}^2}$$

```
public void launch() throws AlgorithmException {
bdim = input.getBDim();
tdim = input.getTDim();
zdim = input.getZDim();
ydim = input.getYDim();
xdim = input.getXDim();
int size = input.size();
Image gradx, grady, gradz, gradt;

switch (operation) {
      case GRADX:
            output = convolve2D(1, 1, 0, 0, 0, new int[][]
                  { { -1, 0, 1 },
                  { -2, 0, 2 }, { -1, 0, 1 } });
                  break;
      case GRADY:
            output = convolve2D(1, 1, 0, 0, 0, new int[][]
                  {{ -1, -2, -1 },
                  { 0, 0, 0 }, { 1, 2, 1 } });
                  break;
      case ORIEN:
            gradx = convolve2D(1, 1, 0, 0, 0, new int[][]
                  { { -1, 0, 1 },
                  { -2, 0, 2 }, { -1, 0, 1 } });
                  grady = convolve2D(1, 1, 0, 0, 0, new int[][]
                  { { -1, -2, -1 },
                  { 0, 0, 0 }, { 1, 2, 1 } });
                  output = new DoubleImage(xdim, ydim, zdim,
            tdim, bdim);
                  output.copyAttributes(input);
                  for (int p = 0; p < size; p++)
                      output.setPixelDouble(p,
                      Math.atan(grady.getPixelDouble(p)
                                          /
gradx.getPixelDouble(p))
                                          - 3 * Math.PI / 4);
            break;
      default:

      }
```

Figure.4.5 - Pseudo-code de traduire l'operateur Sobel en PELICAN

Pour chercher la similarité, nous utilison la même $d(A,B) = \sum_{j=1}^{n} | H_j{}^A - H_j{}^B |$.

Nous calculons les distances entre l'histogramme de l'image requête et les

histogrammes des autres images, le similarité quand trouve les images plus proches.

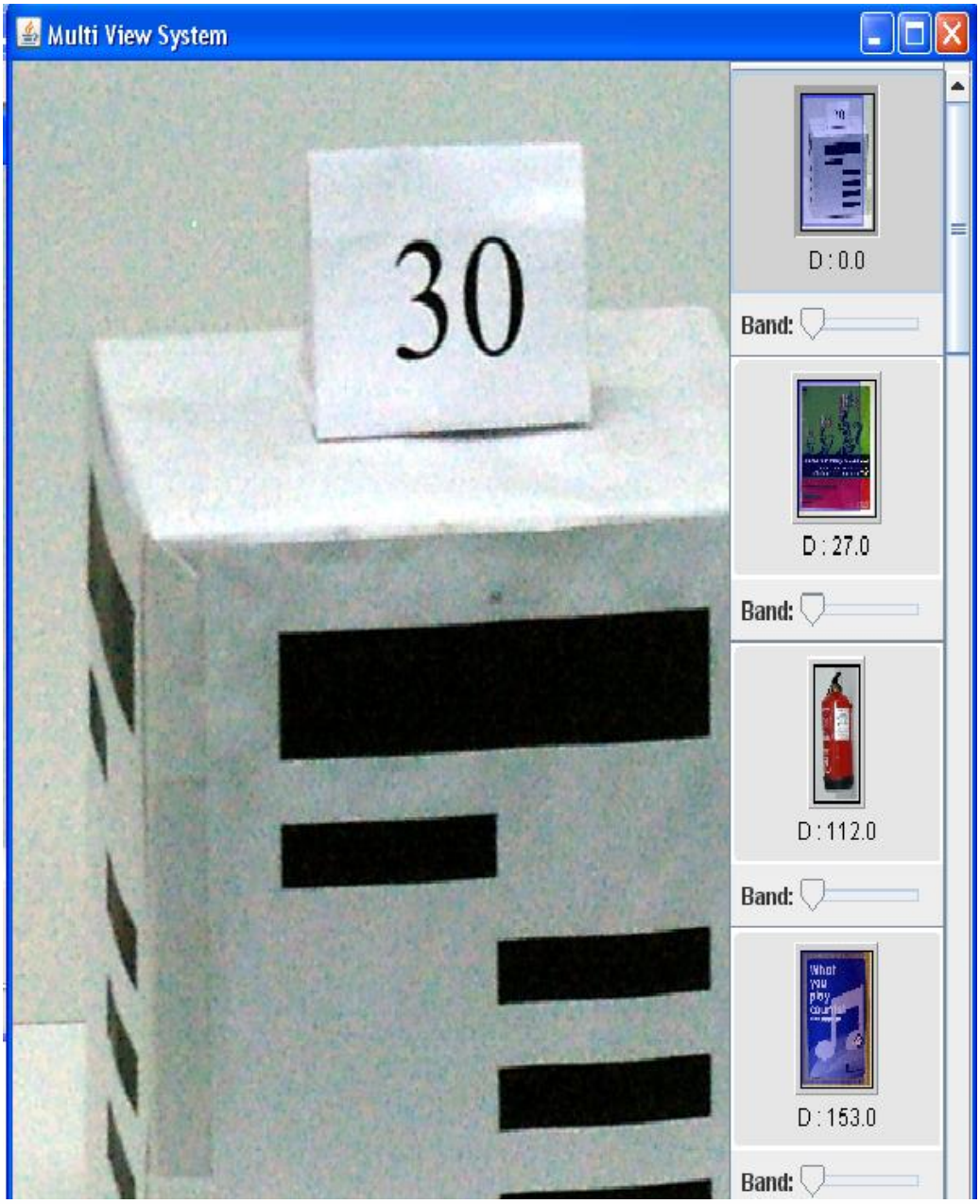


Figure.4.6 - l'application de la similarité d'image basé sur Sobel

4.2. SIFT

4.2.1. Gaussian Pyramid

Tout d'abord, l'image originale (l'image couleur) est converti en niveau de gris avec la bibliothèque plate-forme PELICAN : *RGBToGray.* Cette classe réalise la transformation d'une image RVB tristumulus niveau de gris dans une image en utilisant la formule: g = 0,299 * R + 0,587 * G + 0,114*B. Pour exécuter la classe RGBToGray, les paramètres d'entrée est l'image et les paramètres d'entrée aussi est l'image. Ensuite les deux parameters nous appliquons dans la méthode de convertir l'image couleur en niveau de gris comme indiqué par le tableau.4.1.

Résumé de la méthode	
static Image	exec (Image input) Cette classe réalise la transformation d'une image RVB tristumulus niveau de gris en une image en utilisant la formule: g = 0,299 * R + 0,587 *G + 0,114*B
void	launch ( ) Méthode abstraite pour mettre en œuvre dans chaque algorithme a hérité.

Tableau.4.1 – La méthode de RGBToGray

La figure.10 est le résultat de convertir l'image couleur en niveaux de gris dans le plate-forme PELICAN.

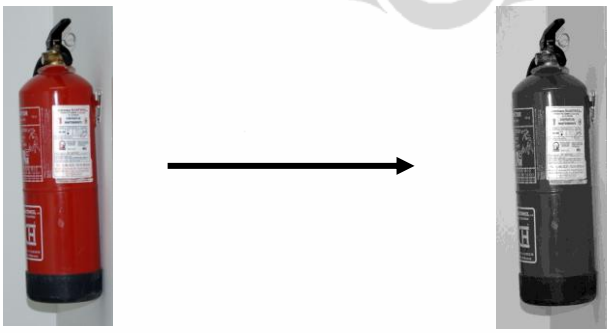


Figure.4.7 - Convertir l'image couleur en niveau de gris

L'image qui a été converti en niveau de gris n'a pas encore défini la valeur de l'échelle. Donc, pour appliquer de filtre gaussienne (en anglais Gaussian Filter) ,il faut faire de certains paramètre σ .

```
kernel=GrayStructuringElement.createSquareFlatStructuringElement(s
ize);
            for (int x = 0; x < kernel.getXDim(); x++) {
                  for (int y = 0; y < kernel.getYDim(); y++) {
                        int _x = x - kernel.getCenter().y;
                        int _y = y - kernel.getCenter().x;
                        kernel.setPixelXYDouble(y, x,
Tools.Gaussian2D(_x, _y, sigma));
                  }
            }
            output = (Image) new Convolution().process(input,
kernel);
      }
```

Figure.4.8 - Le fonction de filtre gaussienne

L'étape suivant, il fault de redimensioner de flou de l'image avant nous créons le Gaussian Pyramid. Dans ce cas, nous utilisons l'interpolation de plus proche voisin (en anglais : nearest neighbor interpolation). Cela grâce à l'interpolation de plus proche voisin est plus simple et rapide à appliquer dans le cas de redimensioner de l'image. L'algorithme de redimensioner de l'image est de trouver place approprié pour mettre les espaces vides à l'intérieur de l'image originale, et de remplir tous les espace avec des couleurs plus vives. Pour la technique l'interpolation de plus proche voisin, les espaces vides seront remplacés par pixel le plus proche voisin. Il résulte une image nette, mais émiettée, et si l'échelle d'agrandir est deux, il semblerait chaque pixel a doublé de taille. Donc, on double la taille de flou de l'image en appliquant la méthode ExpandXY.Interpolation comme indique le tableau.4.2 et après on choisi l'enum de Nearest Interpolation.

Résumé de la méthode	
static ExpandXY.InterpolationMethod	valueOf (java.lang.String name) Retourne la constante enum de ce type avec le nom spécifié
static ExpandXY.InterpolationMethod [ ]	values ( ) Retourne un tableau contenant les constantes de ce type enum, dans l'ordre où elles sont déclarées.

Tableau.4.2 – La méthode de ExpandXY.Interpolation

Le résultat de ce processus va être sauver en tant que le premier image dans Gaussian Pyramid. Pour constuire Pyramide de gradient (en anglais Gaussian Pyramid) il faut appliquer l'image grise avec différent paramètre σ. Dans ce scénario, nous mettons en oeuvre l'échelle, comme le montre le tableau ci-après (la tableau.4.3). Dans cette expérience le Gaussian Pyramid est divisé en 4 octaves et en 5 niveaux de flou l'image, ceci est tel que recommandé par Lowe, inventeur de la méthode SIFT.

Octaves	**l'echelle**				
	0	1	2	3	4
1	0.707107	1.000000	1.414214	2.000000	2.828427
2	1.414214	2.000000	2.828427	4.000000	5.656854
3	2.828427	4.000000	5.656854	8.000000	11.313708
4	5.656854	8.000000	11.313708	16.000000	22.627417

Tableau.4.3 - Différent l'échelle

La taille de l'image chaque octave est la moitié de l'octave précédente.

Résumé champ	
static int	BILINEAR Constant pour représenter méthode bilinéaire: considère la moyenne des pixels X
Image	input L'image d'entrée
static int	NEAREST Constant pour représenter la méthode NEAREST: Considère un pixel pour chaque X
Image	output L'image de sortie
java.lang.Integer	resamplingMethod La méthode de rééchantillonnage : NEAREST, BILINEAR
java.lang.Integer	vb La valeur de ré-échantillonnage sur la dimension B
java.lang.Integer	vt La valeur de ré-échantillonnage sur la dimension T
java.lang.Integer	vx La valeur de ré-échantillonnage sur la dimension X
java.lang.Integer	vy La valeur de ré-échantillonnage sur la dimension Y
java.lang.Integer	vz La valeur de ré-échantillonnage sur la dimension Z

Tableau.4.4 - Reéchantillonnage par valeur

Dans développement de la méthode SIFT, nous définissons les paramètres qui ont le valeur constante comme le montre tableau.4.5.

Parametrages	Le type	Valuer
SIGMA_ANTIALIAS	Double	0.5
SIGMA_PREBLUR	Double	1.0
Octaves	Int	4
Intervals	Int	2
CURVATURE_TRESHOLD	Double	5.0
CONTRAST_TRESHOLD	Double	0.03
NUM_BINS	Int	36
M_PI	Double	3.14159
MAX_KERNEL_SIZE	Int	20
FEATURE_WINDOW_SIZE	Int	16

Tableau.4.5 Parametrage dans SIFT

- SIGMA_ANTIALIAS indique le valeur de flou initial
- SIGMA_PREBLUR indique le valeur de flou après double l'image
- Octaves indique le nombre de l'image dans lequel chaque octave a une taille différente (la taille de l'image de octaves après est la motie de la taille de octave precedent).
- Intervals indique les niveaux de l'image dans chaque octaves qui ont différente échelle.
- CURVATURE_TRESHOLD indique le valeur de seuil de courbure
- CONTRAST_TRESHOLD indique le valeur de seuil de constraste qui est utilisé pour verifier le faible contraste afin que le valuer posséde faible constraste va être supprimé.
- NUM_BINS indique un histogramme d'orientation qui est calculé à partir de l'image gradients autour le point-clé.
- M_PI indique le valeur de phi.
- MAX_KERNEL_SIZE indique le valeur maximum de la réponse impulsionnelle en 2D.
- FEATURE_WINDOW_SIZE indique la zone autour de la point-clé est divisée en 4 * 4 sous-régions.

Cette fonction (figure.14) est la la processus de création Gaussian Pyramid qui, comme décrit ci-dessus, nous définissons un paramètre d'échelle différente. La processus de mise en œuvre de gaussien filtre est fait à partir du premier octave jusqu'à quatrième octave, où dans chaque octave contient cinq images qui ont une échelle différente aussi. En fait, on peut définir l'octave plus ou moins de quatre et on peut également définir de flou l'image plus ou moins de cinq, mais basé sur l'expérience de Lowe, inventeur de la méthode SIFT, quatre octave qui contient cinq images floues sont plus mieux dans que les autres.

```
// Now for the actual image generation
  for (int i = 0; i < m_numOctaves; i++) {
  // Reset sigma for eact octave
  double sigma = initSigma;
  // Get each value image before
  int sizeX = m_gList[i][0].getXDim();
  int sizeY = m_gList[i][0].getYDim();
  int sizeZ = m_gList[i][0].getZDim();
  int sizeB = m_gList[i][0].getBDim();
  int sizeT = m_gList[i][0].getTDim();
  for (int j = 1; j < m_numIntervals + 3; j++) {
  // Calculate a sigma to blur the current image
     to get next one
   double sigma_f = Math.sqrt(Math.pow(2.0, 2.0 /
   m_numIntervals) - 1) *  sigma;
   sigma = Math.pow(2.0, 1.0 / m_numIntervals) * sigma;
   // Store sigma values (to be used later on)
   m_absSigma[i][j] = sigma * 0.5 * Math.pow(2.0, (double) i);
   m_gList[i][j] = GaussianFilter.exec(m_gList[i][j - 1],
   sigma_f);


       m_dogList[i][j - 1] = Difference.exec(m_gList[i][j],
       m_gList[i][j - 1]);
  }

 // If we're not at the las octave
  if (i < m_numOctaves - 1) {
  m_gList    [i + 1][0] = ResamplingByValue.exec(m_gList[i][0],
  sizeX / 2, sizeY / 2, sizeZ, sizeB, sizeT, 0);
   m_absSigma[i + 1][0] = m_absSigma[i][m_numIntervals];
        }
   }
```

Figure.4.9 - Le fonction de création Gaussian Pyramid

La figure 4.10 est le résultat de Gaussian Pyramid qui est mise en oeuvre dans le plate-forme PELICAN.

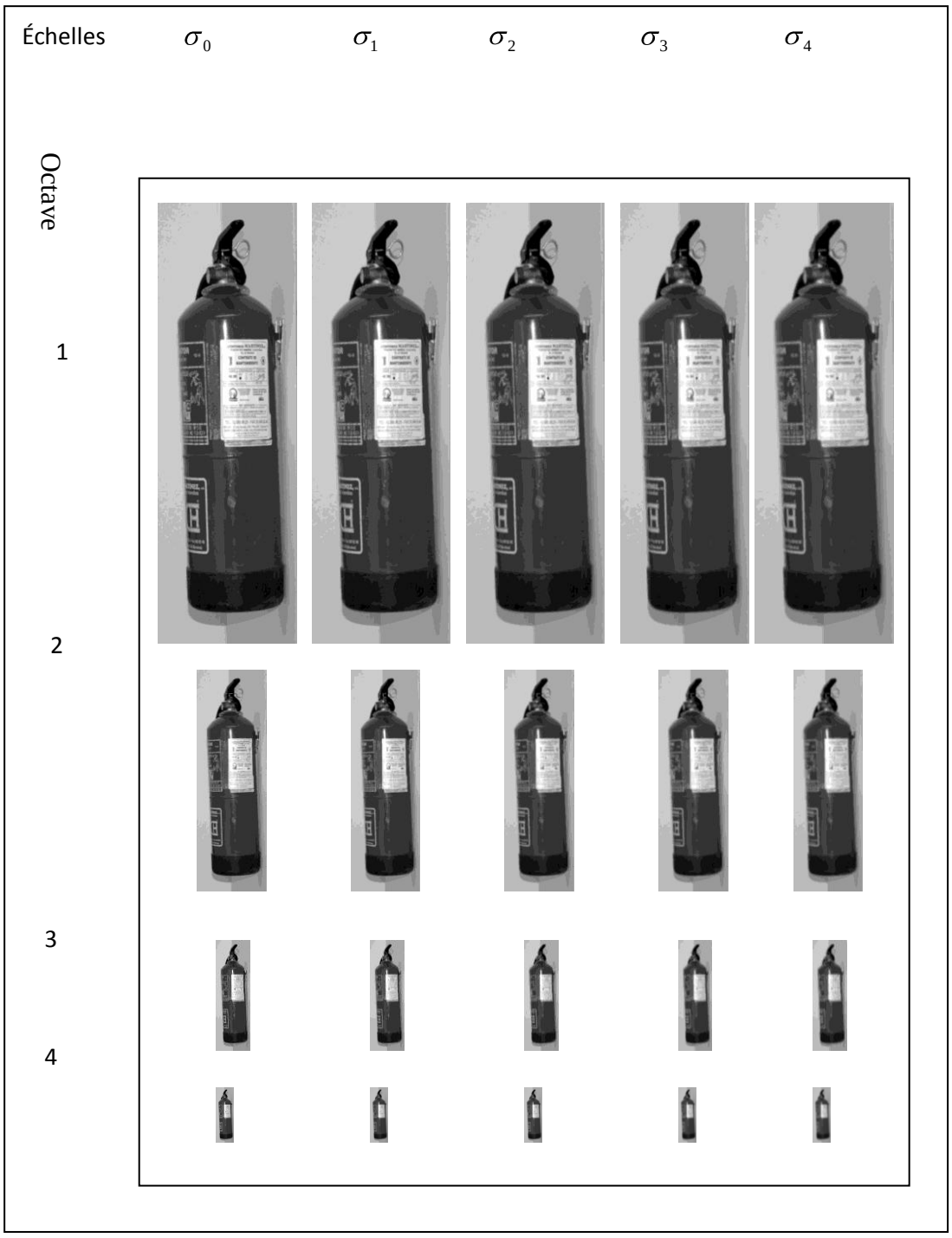


Figure.4.10 - Gaussian Pyramid

4.2.2. DoG (Difference of Gaussian)

Deux images consécutives dans une octave de Gaussian Pyramid sont prélevées et une image est soustraite de l'autre. Alors la paire d'images consécutives suivante est prise, et le processus se se répète. Ceci est fait pour toutes les octaves.

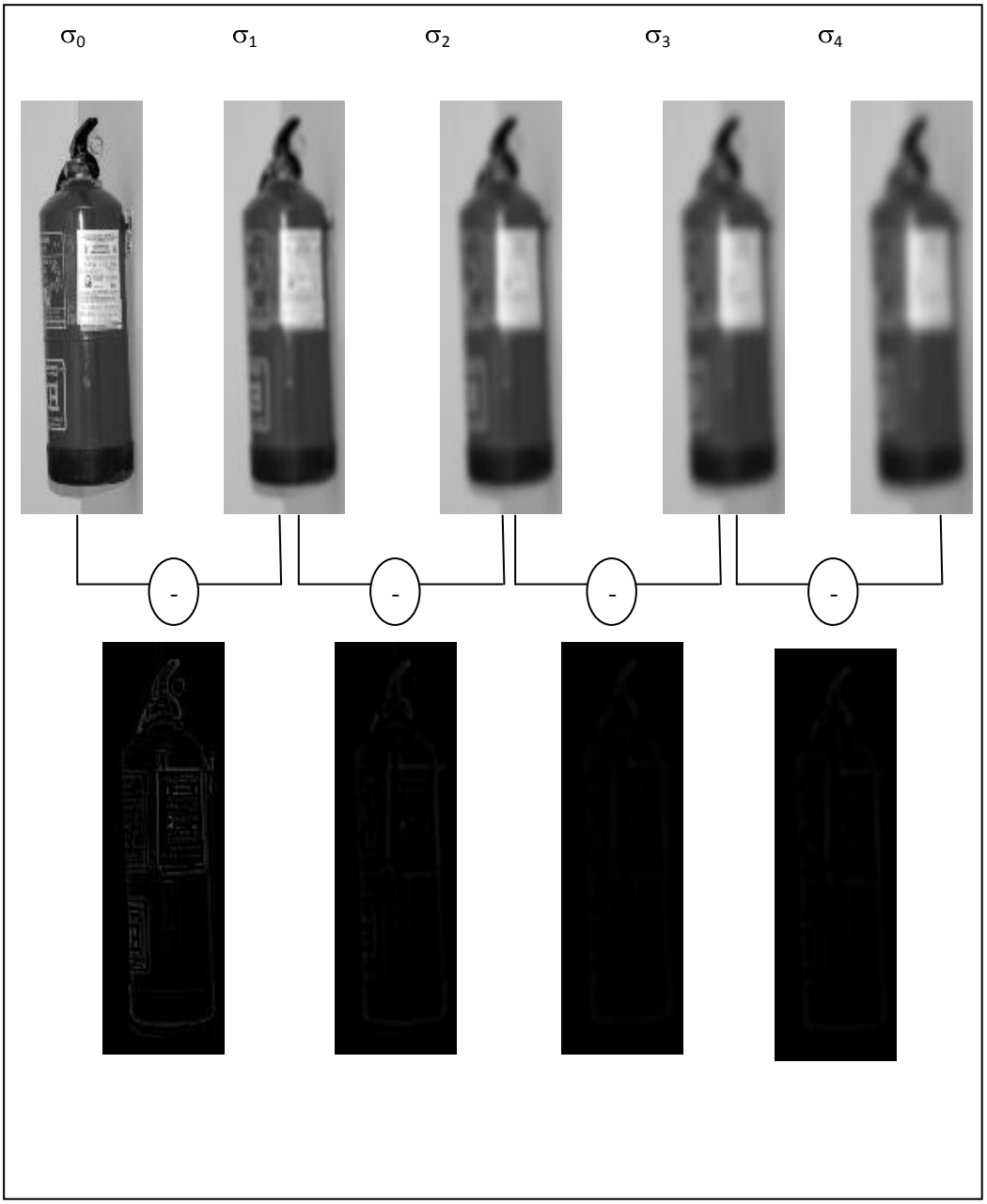


Figure.4.11 – DoG en octave 1

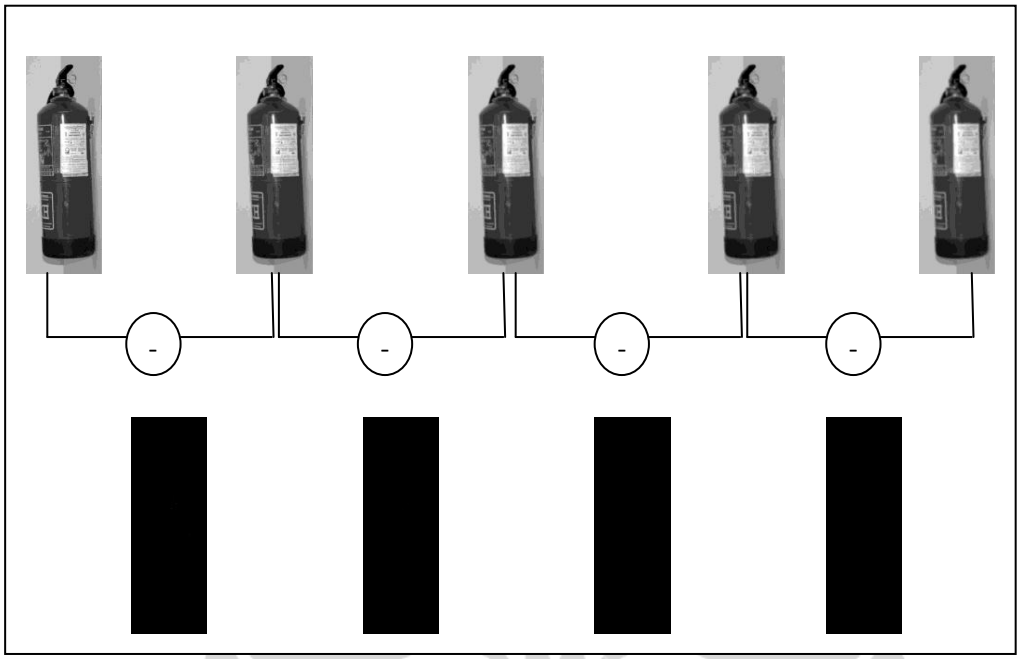


Figure.4.12 – DoG en octave 2

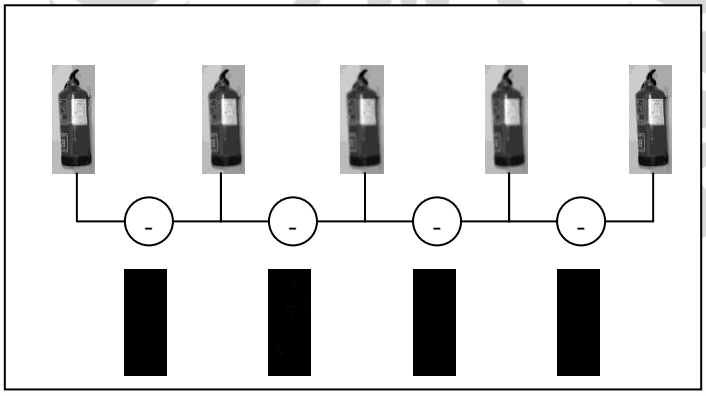


Figure.4.13 – DoG en octave 3

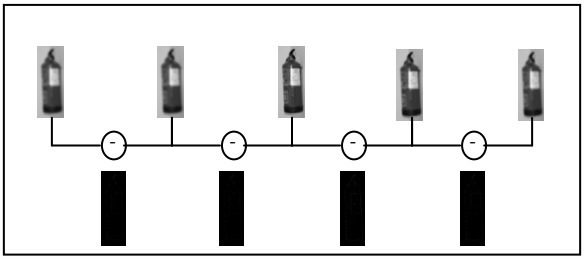


Figure.4.14 – DoG en octave 4

4.2.3. Détection de l'extrema local

Cette technique extrait en premier lieu les points d'intérêt comme des extrêmes locaux de la DOG (Difference of Gaussian) appliquée à différentes échelles. Chaque pixel de l'image filtre DoG est comparé à ses 26 voisins dans le voisinage 3 x 3 x 3 de la pyramide d'images DoG. Un pixel est sélectionné comme un point d'intérêt candidat s'il est un maximum ou minimum local.

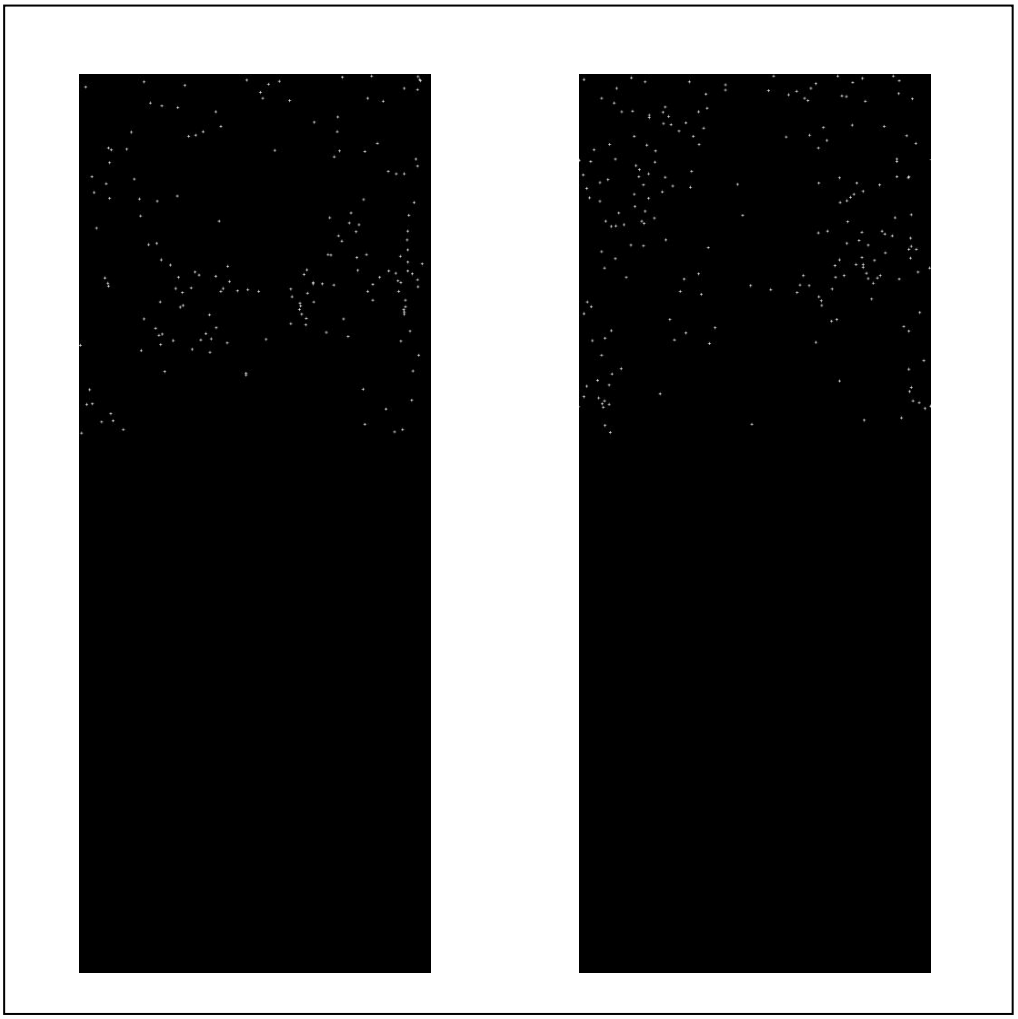


Figure.4.15 - Ensemble des points potentiels

Dans cette image (figure.4.15), nous trouvons 1523 points clés. Cependant, certains d'entre eux sont instables.

4.2.4. Localisation des points clés

Dans ce technique, un seuillage absolu (< |0.03|) sur les D(x) permet d'éliminer les points instables, à faible contraste et les points situés sur les arêtes (ou contours) doivent être éliminés car la fonction DoG y prend des valeurs élevées, ce qui peut donner naissance à des extrema locaux instables.

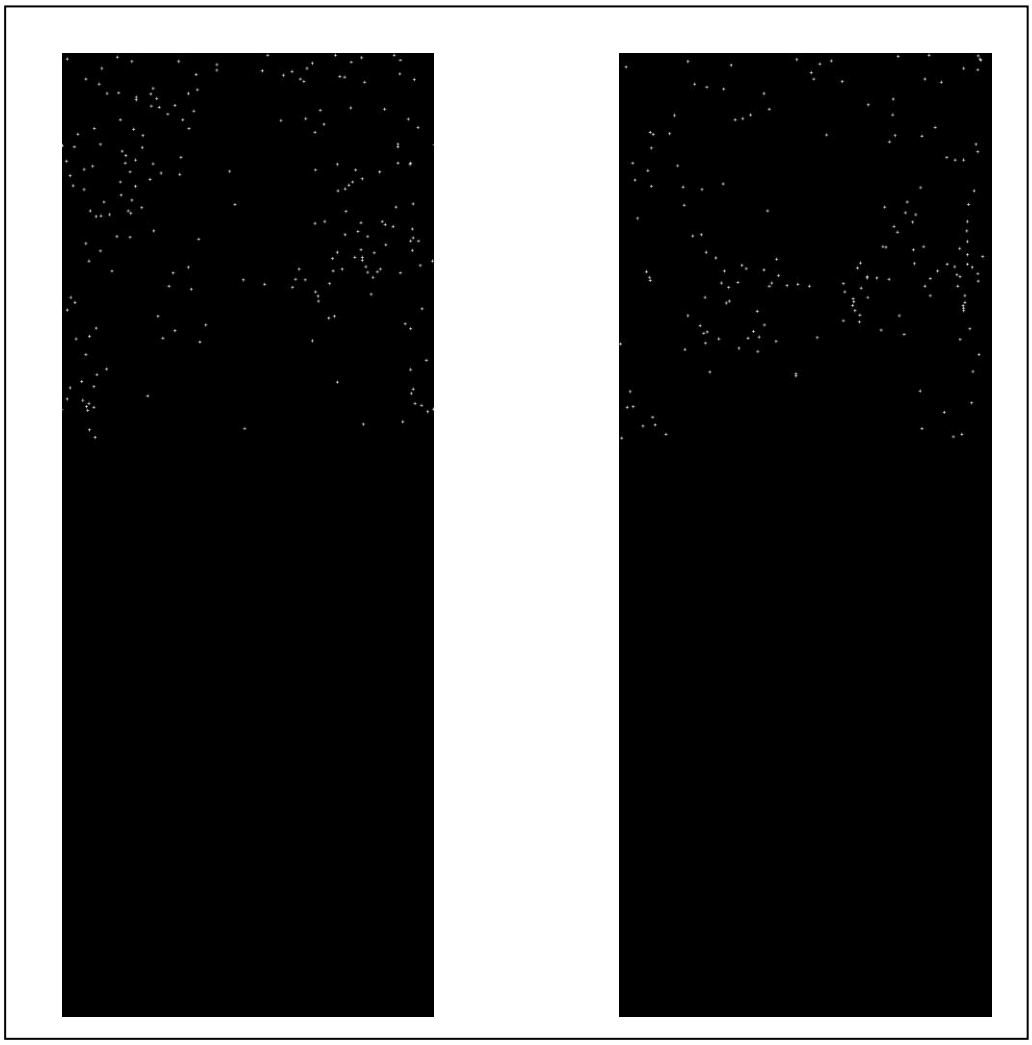


Figure.4.16 - Localisation des points clés

Après de localisation des points clés, nous avons 1518 points clés. Ça veut dire il y a 5 points clés qui a été supprimé. Donc nous pouvons conclure, 5 points clés sont instables.

4.2.6. Descripteur des points clés

Avant nous calculons des vecteurs descripteurs, il faut faire d'assignation d'orientation. L'assignation d'orientation consiste à attribuer à chaque point-clé une ou plusieurs orientations déterminées localement sur l'image à partir de la direction des gradients dans un voisinage autour du point. Dans la mesure où les descripteurs sont calculés relativement à ces orientations, cette étape est essentielle pour garantir l'invariance de ceux-ci à la rotation. À l'issue de ce processus, un point-clé est donc défini par quatre paramètres (x, y, σ, θ). Pour le dernier étape, nous calculons des vecteurs descripteurs. Le dernier résultat de cette étape est représentée par le figure.4.17 suivant :

Figure.4.17 - Descripteur de point-clés

Dans ce cas, le dernier étape , les point-clés située au bord,soit au bord gauche, soit au bord droit. En fait, on a pas bon résultat comme ça. Il faut de quelque point-clés située sur l'objet même s'il y a quelque située au bord.

On test aussi ce algorithme pour l'image de 100 x 100. Dans ces image, les points- clés ne sont pas située sur l'objet mais le processus pour l'image de 100 x 100 est plus rapide que l'image avant, cela à cause de les points- clés ne sont pas beacoup.

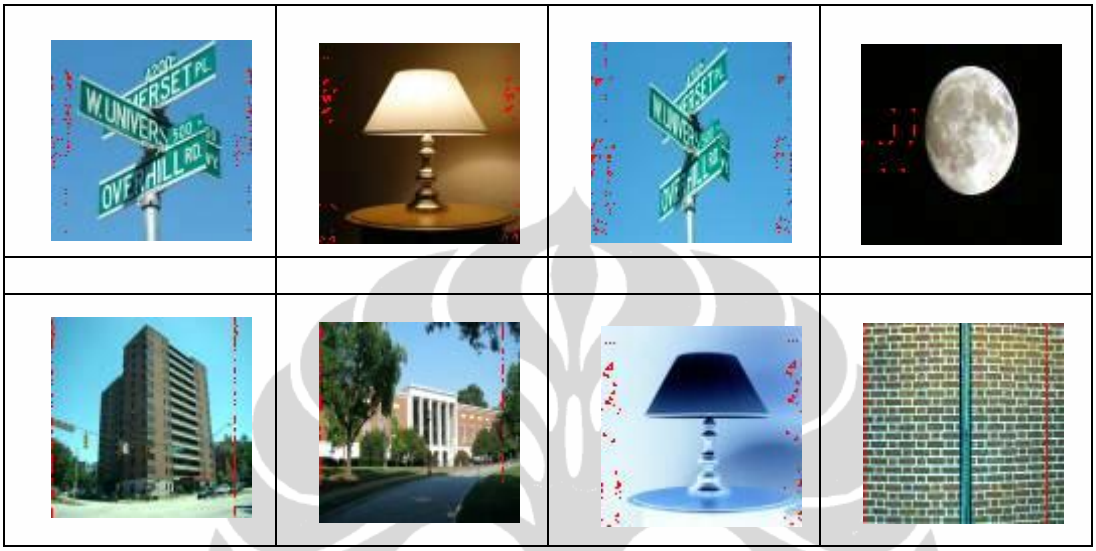


Figure.4.18 - Descripteur de point-clés de quelques images

CHAPITRE 5
CONCLUSION

Au fil de ce travail de fin d'étude, nous avons tenté de présenter un système de recherché d'images par similarité sur l'histogramme d'une image en niveaux de gris et détection de point d'intérêt qui utilise la méthode SIFT (Scale-invariant feature transform), que l'on peut traduire « transformation de caractéristiques visuelles invariante à l'échelle». Lorsque nous avons effectué un stage à IRISA – UBS, nous avons étudié le platforme PELICAN qui permet de concevoir des algorithmes aptes à traiter des types de données variées (l'image 2D ou 3D, à niveaux de gris, couleur, ou multispectrales, séquences d'images, etc.). Ensuite, en utilisant de la plateforme PELICAN, elle constitue désormais une expérience professionnelle valorisante et encourageante pour mon avenir dans le domaine traitement d'images.

Lorsque nous mettons en œuvre la méthode SIFT qui utilise la bibliothèque PELICAN, si l'image génére beaucoup de point clés, le processus de calcul de descripteur de point clés est lent (nous utilisons l'image de 307 x 768). Mais si l'image qui a petite taille, par exemple : l'image de 100 x 100, le processus de calcul de descripteur de point clés est rapide.

Références

1) Site IRISA – UBS
   http://www-irisa.univ-ubs.fr/index.shtml.fr

2) Site Équipe SEASIDE
   http://www-valoria.univ-ubs.fr/SEASIDE/index.shtml.fr

3) Site de Recherce d'image par le contenu
   http://fr.wikipedia.org/wiki/Recherche_d'image_par_le_contenu

4) Site de SIFT. En ligne. URL :
   http://fr.wikipedia.org/wiki/Scale-invariant_feature_transform

5) Sébastien Lefèvre, "PELICAN : une plate-forme pour l'analyse et le traitement des images", VALORIA / IRISA-UBS (équipe SEASIDE), Vannes le 7 Novembre 2011.

6) LÊ THI LAN, "Indexation et Rescherche d'images par le contenu", Mémoire de Master, Hanoi 2005.

7) Florent Sollier, "PELICAN : Une architecture générique", Mémoire de Master 2 Ingénierie Logicielle de Université Louis Pasteur, Strasbourg 2008.

8) HOANG Thanh Lam, "Indexation et Recherche d'images par l'utisation de mots visuels", ", Mémoire de Master de l'Institut de la Franchophonie pour l'Informatique, Hanoi 2008.

9) David G.Lowe, "Distinctive Image Feature from Scale-invariant Keypoints", Computer Science Departement-University of British Columbia, Vancouver le 5 Janvier 2004.
10) Lahouli Ichraf, "Classification des lieux dans un milieu indoor en utilisant des detecteurs 2D et 3D", Ecole Royale Militaire Bruxelles.
11) Site SIFT et SURF
    http://www.olivier-augereau.com/blog/?p=73

L A M P I R A N

Code SIFT dans le plateforme PELICAN

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package fr.unistra.pelican.algorithms.ichsan.siftpoint;

import Jama.Matrix;
import fr.unistra.pelican.Algorithm;
import fr.unistra.pelican.AlgorithmException;
import fr.unistra.pelican.ByteImage;
import fr.unistra.pelican.Image;
import fr.unistra.pelican.algorithms.arithmetic.Difference;
import fr.unistra.pelican.algorithms.conversion.RGBToGray;
import fr.unistra.pelican.algorithms.draw.DrawCircle;
import fr.unistra.pelican.algorithms.draw.DrawCircleAndres;
import fr.unistra.pelican.algorithms.draw.DrawCircleBresenham;
import fr.unistra.pelican.algorithms.draw.DrawCross;
import fr.unistra.pelican.algorithms.geometric.ExpandXY;
import fr.unistra.pelican.algorithms.geometric.ResamplingByValue;
import fr.unistra.pelican.algorithms.io.ImageLoader;
import fr.unistra.pelican.algorithms.morphology.gray.GrayCornerDetection;
import fr.unistra.pelican.algorithms.spatial.GaussianFilter;
import fr.unistra.pelican.algorithms.visualisation.MViewer;
import fr.unistra.pelican.algorithms.visualisation.Viewer2D;
import fr.unistra.pelican.gui.MultiViews.MultiView;
import fr.unistra.pelican.util.Line;
import java.awt.Color;
import java.awt.Point;
import java.io.File;
import java.util.ArrayList;
import java.util.Vector;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

/**
 *
 * @author kampus90
 */
public class SIFTrev extends Algorithm {

    /**
     * Input parameter.
     */
    public static Image input;
    public static double SIGMA_ANTIALIAS = 0.5;
    public static double SIGMA_PREBLUR = 1.0;
    public static int m_numOctaves = 4;
    public static int m_numIntervals = 2;
    public static int Octaves = 4;
    public static int Intervals = 2;
    public static final double CURVATURE_THRESHOLD = 5.0;
    public static final double CONTRAST_TRESHOLD = 0.03;
    public static final int NUM_BINS = 36;
    public static final double M_PI = 3.14159265358979323846264338327795;
    public static final int MAX_KERNEL_SIZE = 20;
    public static final int FEATURE_WINDOW_SIZE = 16;
    public static final int DESC_NUM_BINS = 8;
```

```
    public static int m_numKeypoints;
    public static int FVSIZE = 128;
    public static double FV_THRESHOLD = 0.2;
    public static Vector<Double> orien = new Vector<Double>();
    public static Vector<Double> mag = new Vector<Double>();
    public static Image temp;
    /**
     * Declartion Gaussian Pyramid
     */
    public static Image[][] m_gList = new Image[Octaves][Intervals + 3];
    /**
     * Declaration of Differefe of Gaussian
     */
    public static Image[][] m_dogList = new Image[Octaves][Intervals + 2];
    public static Vector<Keypoint3> m_keyPoints = new Vector<Keypoint3>();
    public static double[][] m_absSigma = new double[Octaves][Intervals + 3];
    /**
     * Output parameter.
     */
    // public Image output;
    public static Image output;


    /**
     * Constructor
     *
     */
    public SIFTrev() {
        super.inputs = "input";
        super.outputs = "output";

    }

    public static Image exec(Image input) {
        return (Image) new SIFTrev().process(input);
    }


    public void launch() throws AlgorithmException {

        // Declaration
        Image[][] BuildScaleSpace = new Image[Octaves][Intervals + 2];
        Image[][] DetectingExtrema = new Image[Octaves][Intervals + 2];
        Image[][] AssignOrientations = new Image[Octaves][Intervals];
        Image ExtractKeypointDescriptors;;

        // Call method
BuildScaleSpace = Create_Scale_Space(input, Octaves, Intervals + 2);
DetectingExtrema = Create_Extrema(BuildScaleSpace, Octaves, Intervals);
AssignOrientations = build_orientation(DetectingExtrema, m_absSigma, m_gList,
input, Octaves, Intervals, m_keyPoints);
ExtractKeypointDescriptors = build_descriptors(input, temp, m_gList, Octaves,
Intervals, m_keyPoints);
output = ExtractKeypointDescriptors;

}
```

```
 /**
  * Method for creating Gaussian Pyramid and DOG
  *
  * @param src Source Image
  * @param Octaves The number Octaves =4
  * @param Intervals The number Intervals =2
  * @return DoG (List of Difference Gaussian)
  */
private Image[][] Create_Scale_Space(Image input, int Octaves, int Intervals) {

        Image imGray = RGBToGray.exec(input);
        temp = imGray.copyImage(true);


        //Do blur the image with a sigma 0.5
        Image ImgBlur = GaussianFilter.exec(imGray, SIGMA_ANTIALIAS);

        // Create an image double the dimension
        int xdim = imGray.getXDim();
        int ydim = imGray.getYDim();
        m_gList[0][0] = ExpandXY.exec(ImgBlur, xdim * 2, ydim * 2,
        ExpandXY.InterpolationMethod.Nearest);

        // Preblur this base image
        m_gList[0][0] = GaussianFilter.exec(m_gList[0][0], SIGMA_PREBLUR);



        double initSigma = Math.sqrt(2.0);

        // Keep a track of the sigma
        m_absSigma[0][0] = initSigma * 0.5;

        // Now for the actual image generation
        for (int i = 0; i < m_numOctaves; i++) {
            // Reset sigma for eact octave
            double sigma = initSigma;
            // Get each value image before
            int sizeX = m_gList[i][0].getXDim();
            int sizeY = m_gList[i][0].getYDim();
            int sizeZ = m_gList[i][0].getZDim();
            int sizeB = m_gList[i][0].getBDim();
            int sizeT = m_gList[i][0].getTDim();
            //System.out.println("Nilai currentsize"+currentSize);
            for (int j = 1; j < m_numIntervals + 3; j++) {
                // Calculate a sigma to blur the current image to get next one
                double sigma_f = Math.sqrt(Math.pow(2.0, 2.0 / Intervals) - 1)
                * sigma;
                sigma = Math.pow(2.0, 1.0 / m_numIntervals) * sigma;
                // Store sigma values (to be used later on)
                m_absSigma[i][j] = sigma * 0.5 * Math.pow(2.0, (double) i);

                //Apply gaussian filter
                m_gList[i][j] = GaussianFilter.exec(m_gList[i][j - 1], sigma_f);

                // Calculate the DoG Image
```

```
                m_dogList[i][j - 1] = Difference.exec(m_gList[i][j],
                m_gList[i][j - 1]);
            }

            // If we're not at the las octave
            if (i < m_numOctaves - 1) {
            m_gList    [i + 1][0]    = ResamplingByValue.exec(m_gList[i][0],
            sizeX / 2, sizeY / 2, sizeZ, sizeB, sizeT, 0);
                m_absSigma[i + 1][0]    = m_absSigma[i][m_numIntervals];
            }
        }

        return m_dogList;
    }

    /**
     * Method for Detecting Extrema and Minima
     *
     * @param BuildScaleSpace is Difference Gaussian Pyramid
     * @param Octaves The number of Octaves = 4
     * @param Intervals The number of Intervals = 2
     * @return An image with localization point
     */
    private static Image[][] Create_Extrema(Image[][] BuildScaleSpace, int
    Octaves, int Intervals) {

        System.out.println("Detecting extrema");
        // Looping variables
        int i, j, xi, yi;

        // some variabel we'll use later on
        double curvature_ratio, curvature_threshold;
        Image midle, up, down;
        int scale;
        Image[][] m_extrema = new Image[4][2];

        double dxx, dyy, dxy, trH, detH;


        int num = 0;            // Number of keypoins detected
        int numRemoved = 0;     // The number of key points rejected
        curvature_threshold = (CURVATURE_THRESHOLD + 1) * (CURVATURE_THRESHOLD
        + 1) / CURVATURE_THRESHOLD;

        int[] cscolor = {255, 0, 0};    // points crosses color
        int[] black = {0,0,0};
        int crosssize = 1;
        Point p1, p2, p3, p4;
        int[] lkcolor = {0, 0, 255};    // links between points color

        // Detect extrema in the DoG images
        for (i = 0; i < Octaves; i++) {
            scale = (int) Math.pow(2.0, (double) i);
            for (j = 1; j < Intervals + 1; j++) {
                // Copy DOG to extrema
                m_extrema[i][j - 1] = BuildScaleSpace[i][0].copyImage(true);
```

```
// Image just above and below, in the current octave
midle = GrayCornerDetection.exec(BuildScaleSpace[i][j]);
up    = GrayCornerDetection.exec(BuildScaleSpace[i][j+1]);
down  = GrayCornerDetection.exec(BuildScaleSpace[i][j-1]);

for (xi = 1; xi < BuildScaleSpace[i][j].getXDim() - 1; xi++) {
    for (yi = 1; yi < BuildScaleSpace[i][j].getYDim() - 1; yi++)
{
        // true if a keypoint is a maxima/minima
        boolean justSet     = false;

        double currentPixel = midle.getPixelXYByte(xi, yi);


        // Check for a maximum
if (currentPixel > midle.getPixelXYByte(xi, yi - 1)
    && currentPixel > midle.getPixelXYByte(xi, yi + 1)
    && currentPixel > midle.getPixelXYByte(xi - 1, yi)
    && currentPixel > midle.getPixelXYByte(xi + 1, yi)
    && currentPixel > midle.getPixelXYByte(xi - 1, yi - 1)
    && currentPixel > midle.getPixelXYByte(xi + 1, yi - 1)
    && currentPixel > midle.getPixelXYByte(xi + 1, yi + 1)
    && currentPixel > midle.getPixelXYByte(xi - 1, yi + 1)
    && currentPixel > up.getPixelXYByte(xi, yi)
    && currentPixel > up.getPixelXYByte(xi, yi - 1)
    && currentPixel > up.getPixelXYByte(xi, yi + 1)
    && currentPixel > up.getPixelXYByte(xi - 1, yi)
    && currentPixel > up.getPixelXYByte(xi + 1, yi)
    && currentPixel > up.getPixelXYByte(xi - 1, yi - 1)
    && currentPixel > up.getPixelXYByte(xi + 1, yi - 1)
    && currentPixel > up.getPixelXYByte(xi + 1, yi + 1)
    && currentPixel > up.getPixelXYByte(xi - 1, yi + 1)
    && currentPixel > down.getPixelXYByte(xi, yi)
    && currentPixel > down.getPixelXYByte(xi, yi - 1)
    && currentPixel > down.getPixelXYByte(xi, yi + 1)
    && currentPixel > down.getPixelXYByte(xi - 1, yi)
    && currentPixel > down.getPixelXYByte(xi + 1, yi)
    && currentPixel > down.getPixelXYByte(xi - 1, yi - 1)
    && currentPixel > down.getPixelXYByte(xi + 1, yi - 1)
    && currentPixel > down.getPixelXYByte(xi + 1, yi + 1)
    && currentPixel > down.getPixelXYByte(xi - 1, yi + 1)) {

            p1 = new Point(yi, xi);
            drawCross(m_extrema[i][j-1], p1,
            crosssize,cscolor);
            num++;
            justSet = true;

            System.out.println("The number of Keypoint maxima ="
            + num);


        } // Check for minima
     else if (currentPixel   < midle.getPixelXYByte(xi, yi - 1)
     && currentPixel < midle.getPixelXYByte(xi, yi + 1)
     && currentPixel < midle.getPixelXYByte(xi - 1, yi)
      && currentPixel < midle.getPixelXYByte(xi + 1, yi)
```

```
&& currentPixel < midle.getPixelXYByte(xi - 1, yi - 1)
&& currentPixel < midle.getPixelXYByte(xi + 1, yi - 1)
&& currentPixel < midle.getPixelXYByte(xi + 1, yi + 1)
&& currentPixel < midle.getPixelXYByte(xi - 1, yi + 1)
&& currentPixel < up.getPixelXYByte(xi, yi)
&& currentPixel < up.getPixelXYByte(xi, yi - 1)
&& currentPixel < up.getPixelXYByte(xi, yi + 1)
&& currentPixel < up.getPixelXYByte(xi - 1, yi)
&& currentPixel < up.getPixelXYByte(xi + 1, yi)
&& currentPixel < up.getPixelXYByte(xi - 1, yi - 1)
&& currentPixel < up.getPixelXYByte(xi + 1, yi - 1)
&& currentPixel < up.getPixelXYByte(xi + 1, yi + 1)
&& currentPixel < up.getPixelXYByte(xi - 1, yi + 1)
&& currentPixel < down.getPixelXYByte(xi, yi)
&& currentPixel < down.getPixelXYByte(xi, yi - 1)
&& currentPixel < down.getPixelXYByte(xi, yi + 1)
&& currentPixel < down.getPixelXYByte(xi - 1, yi)
&& currentPixel < down.getPixelXYByte(xi + 1, yi)
&& currentPixel < down.getPixelXYByte(xi - 1, yi - 1)
&& currentPixel < down.getPixelXYByte(xi + 1, yi - 1)
&& currentPixel < down.getPixelXYByte(xi + 1, yi + 1)
&& currentPixel < down.getPixelXYByte(xi - 1, yi + 1)) {

        p2 = new Point(yi, xi);

        drawCross(m_extrema[i][j-1], p2,
        crosssize,cscolor);
        num++;
        justSet = true;

        System.out.println("The number of Keypoint minima ="
        + num);
    }

    // The contrast check
    if (justSet &&
    Math.abs(midle.getPixelXYByte(xi, yi)) <
    CONTRAST_TRESHOLD) {

        p3 = new Point(yi, xi);
        drawCross(m_extrema[i][j-1], p3, crosssize,black);
        num--;
        numRemoved++;

        justSet = false;
    }

    // The edge check
    if (justSet) {
        dxx = midle.getPixelXYByte(xi, yi - 1)
                + midle.getPixelXYByte(xi, yi + 1)
                - 2 * midle.getPixelXYByte(xi, yi);

        dyy =   midle.getPixelXYByte(xi - 1, yi)
                + midle.getPixelXYByte(xi + 1, yi)
                - 2 * midle.getPixelXYByte(xi, yi);
```

```
            dxy = (midle.getPixelXYByte(xi - 1, yi - 1)
                  + midle.getPixelXYByte(xi + 1, yi + 1)
                  - midle.getPixelXYByte(xi + 1, yi - 1)
                  - midle.getPixelXYDouble(xi - 1, yi + 1)) / 4.0;

                  trH = dxx + dyy;
                  detH = dxx * dyy - dxy * dxy;

                  curvature_ratio = trH * trH / detH;
                  if (detH < 0 || curvature_ratio > curvature_threshold) {
                     p4 = new Point(yi, xi);
                     drawCross(m_extrema[i][j-1], p4, crosssize,cscolor);

                     num--;
                     numRemoved++;

                     justSet = false;

                           }
                        }

                     }
                  }

               }
            }

         m_numKeypoints = num;
         System.out.println("The number of Keypoint =" + num);
         System.out.println("The number of Rejected =" + numRemoved);
         return m_extrema;
      }

private static Image[][] build_orientation(Image[][] DetectingExtrema,
double[][] m_absSigma, Image[][] m_gList, Image src, int Octaves, int Intervals,
Vector<Keypoint3> m_keyPoints) {

         System.out.println("Assigning Orientations");
         int i, j, k, xi, yi;
         int kk, tt;
         Image[][] magnitude = new Image[Octaves][Intervals];
         Image[][] orientation = new Image[Octaves][Intervals];

         for (i = 0; i < Octaves; i++) {
         // Iterate through all scales
         for (j = 1; j < Intervals + 1; j++) {
         // Get image from Gaussian Pyramid
                magnitude[i][j - 1]   = m_gList[i][j].copyImage(true);
                orientation[i][j - 1] = m_gList[i][j].copyImage(true);


                // Iterate over the gausian image with current octave and interval
                for (xi = 1; xi < m_gList[i][j].getXDim() - 1; xi++) {
                    for (yi = 1; yi < m_gList[i][j].getYDim() - 1; yi++) {
                        // Calculate Gradient
                        double dx = m_gList[i][j].getPixelXYDouble(xi + 1, yi)
                             - m_gList[i][j].getPixelXYDouble(xi - 1, yi);
```

```
                        double dy = m_gList[i][j].getPixelXYDouble(xi, yi + 1)
                                - m_gList[i][j].getPixelXYDouble(xi, yi - 1);

    // Store magnitude
    magnitude[i][j - 1].setPixelXYDouble(xi, yi, Math.sqrt(dx * dx + dy * dy));

   // Store Orientation
   double ari = (Math.atan2(dy, dx) == M_PI) ? -M_PI : Math.atan2(dy, dx);

                        orientation[i][j - 1].setPixelXYDouble(xi, yi, ari);

                    }
                }
            }
        }

        // The histogram with 8 bins
        double[] hist_orient = new double[NUM_BINS];

        // Go through all octaves
        for (i = 0; i < Octaves; i++) {
            // Store current scale, width and height
            int scale = (int) Math.pow(2.0, (double) i);
            int width = m_gList[i][0].getXDim();
            int height = m_gList[i][0].getYDim();

            // Go through all intervals in the current scale
            for (j = 1; j < Intervals + 1; j++) {
                double abs_sigma = m_absSigma[i][j];

                // This is used for magnitudes
                Image imgWeight;
                imgWeight = GaussianFilter.exec(magnitude[i][j - 1], 1.5 *
                abs_sigma);

                // Get the kernel size for Gaussian blur
                int hfsz = GetKernelSize(1.5 * abs_sigma) / 2;

                // Temporarily used to generate a mask of region used to calculate
                // the orientations
                Image imgMask = src.copyImage(true);

                // Iterate through all points at this octave and interval
                for (xi = 0; xi < width; xi++) {
                    for (yi = 0; yi < height; yi++) {
                        // We're at a keypoint
                            if (DetectingExtrema[i][j - 1].getPixelXYDouble(xi,
                            yi) != 0) {
                            // Reset the histogram thingy
                            for (k = 0; k < NUM_BINS; k++) {
                                hist_orient[k] = 0.0;
                            }

                            // Go through all pixel in the window around the
                            extrema
for (kk = -hfsz; kk <= hfsz; kk++) {
```

```
for (tt = -hfsz; tt <= hfsz; tt++) {
// Ensure we're within the image
 if (xi + kk < 0 || xi + kk >= width || yi + tt < 0 || yi + tt >= height)
 {

     continue;
 }

 double sampleOrient = orientation[i][j - 1].getPixelXYDouble(xi + kk, yi
 + tt);

 if (sampleOrient <= -M_PI || sampleOrient > M_PI) {
     System.out.println("Bad Orientation: " + sampleOrient);
 }

 sampleOrient += M_PI;

 // Convert to degrees
 int sampleOrientDegrees = (int) (sampleOrient * 180 / M_PI);
 hist_orient[(int) sampleOrientDegrees / (360 / NUM_BINS)] +=
 imgWeight.getPixelXYDouble(xi + kk, yi + tt);
      }
 }

 // We've computed the histogram. Now check for the maximum
 double max_peak = hist_orient[0];
 int max_peak_index = 0;
 for (k = 1; k < NUM_BINS; k++) {
      if (hist_orient[k] > max_peak) {
          max_peak = hist_orient[k];
          max_peak_index = k;
      }
 }

 for (k = 0; k < NUM_BINS; k++) {
 // Do we have a good peak?
 if (hist_orient[k] > 0.8 * max_peak) {


// Three points. (x2,y2) is the peak and (x1,y1)
// and (x3,y3) are the neigbours to the left and right.
// If the peak occurs at the extreme left, the "left
// neighbour" is equal to the right most. Similarly for
// the other case (peak is rightmost)
double x1 = k - 1;
double y1;
double x2 = k;
double y2 = hist_orient[k];
double x3 = k + 1;
double y3;
if (k == 0) {
y1 = hist_orient[NUM_BINS - 1];
y3 = hist_orient[1];
} else if (k == NUM_BINS - 1) {
y1 = hist_orient[NUM_BINS - 2];
y3 = hist_orient[0];
} else {
```

```
        y1 = hist_orient[k - 1];
        y3 = hist_orient[k + 1];
        }
        // Next we fit a downward parabola aound
        // these three points for better accuracy
        // A downward parabola has the general form
        //
        // y = a * x^2 + bx + c
        // Now the three equations stem from the three points
        // (x1,y1) (x2,y2) (x3.y3) are
        //
        // y1 = a * x1^2 + b * x1 + c
        // y2 = a * x2^2 + b * x2 + c
        // y3 = a * x3^2 + b * x3 + c
        //
        // in Matrix notation, this is y = Xb, where
        // y = (y1 y2 y3)' b = (a b c)' and
        //
        //     x1^2 x1 1
        // X = x2^2 x2 1
        //     x3^2 x3 1
        //
        // OK, we need to solve this equation for b
        // this is done by inverse the matrix X
        //
        // b = inv(X) Y

        double[] b = new double[3];
        Matrix X = new Matrix(3, 3);
        Matrix matInv = new Matrix(3, 3);

        X.set(0, 0, x1 * x1);
        X.set(0, 1, x1);
        X.set(0, 2, 1);
        X.set(1, 0, x2 * x2);
        X.set(1, 1, x2);
        X.set(1, 2, 1);
        X.set(2, 0, x3 * x3);
        X.set(2, 1, x3);
        X.set(2, 2, 1);
        // Invert the matrix
        X.inverse();
        matInv =X;

b[0] = matInv.get(0, 0) * y1 + matInv.get(0, 1) * y2 + matInv.get(0, 2) * y3;
b[1] = matInv.get(1, 0) * y1 + matInv.get(1, 1) * y2 + matInv.get(1, 2) * y3;
b[2] = matInv.get(2, 0) * y1 + matInv.get(2, 1) * y2 + matInv.get(2, 2) * y3;

// the vertex of parabola is (-b/(2a),c-b^2/4a)
double x0 = -b[1] / (2 * b[0]);
// Anomalous situation
if (Math.abs(x0) > 2 * NUM_BINS) {
    x0 = x2;
}
// make sure it is within the range
while (x0 < 0) {
x0 += NUM_BINS;
```

```
}
while (x0 >= NUM_BINS) {
       x0 -= NUM_BINS;
}
// Normalize it
double x0_n = x0 * (2 * M_PI / NUM_BINS);
assert (x0_n >= 0 && x0_n < 2 * M_PI);
        x0_n -= M_PI;
assert (x0_n >= -M_PI && x0_n < M_PI);
       orien.add(x0_n);
       mag.add(hist_orient[k]);
      }
}
// Save this keypoint into the list
m_keyPoints.add(Keypoint3((xi * scale) / 2, (yi * scale) / 2, mag, orien, i *
Intervals + j - 1));

             }

          }
        }
     }
 }

 return orientation;

}
private static int GetKernelSize(double sigma) {
int i;
double cut_off = 0.001;
for (i = 0; i < MAX_KERNEL_SIZE; i++) {
    if (Math.exp(-((double) (i * i)) / (2.0 * sigma * sigma)) < cut_off) {
        break;
    }
  }
int size = 2 * i - 1;
return size;
}

private static Keypoint3 Keypoint3(int x, int y, Vector<Double> m, Vector<Double>
o, int s) {
float xi;
float yi;   // It's location
Vector<Double> mag; // The list of magnitudes at this point
Vector<Double> orien;   // The list of orientations detected
int scale;  // The scale where this was detect

xi = x;
yi = y;
mag = m;
orien = o;
scale = s;
return new Keypoint3(xi, yi, mag, orien, scale);
}

private static Image build_descriptors(Image src, Image temp, Image[][] m_gList,
int Octaves, int Intervals, Vector<Keypoint3> m_keyPoints) {
```

```
System.out.println("Extract Keypoint descriptors\n");
// For loops
int i, j;

        // Interpolated thingy. We're dealing with "inbetween" gradient
        // magnitudes and orientations
        Image[][] imgInterpolatedMagnitude   = new Image[Octaves][Intervals];
        Image[][] imgInterpolatedOrientation = new Image[Octaves][Intervals];

        // These two loops calculate the interpolated thingy for all octaves
        // and subimages

        for (i = 0; i < Octaves; i++) {
            for (j = 1; j < Intervals + 1; j++) {
                int width = m_gList[i][j].getXDim();
                int height = m_gList[i][j].getYDim();
                int zdim = m_gList[i][j].getZDim();
                int bdim = m_gList[i][j].getBDim();
                int tdim = m_gList[i][j].getTDim();
                // Create an Image and zero it out
Image imgTemp = ResamplingByValue.exec(m_gList[i][j], width * 2, height * 2, zdim,
bdim, tdim, 0);


// Copy Gaussian Pyramid
imgTemp = m_gList[i][j].copyImage(true);

// Allocate memory
imgInterpolatedMagnitude[i][j - 1] = ResamplingByValue.exec(m_gList[i][j], width
+ 1, height + 1, zdim, bdim, tdim, 0);
imgInterpolatedOrientation[i][j - 1] = ResamplingByValue.exec(m_gList[i][j],
width + 1, height + 1, zdim, bdim, tdim, 0);


// Do the calculations
for (int ii = (int) 1.5; ii < width - 1.5; ii++) {
     for (int jj = (int) 1.5; jj < height - 1.5; jj++) {
    // "inbetween" change

double dx = m_gList[i][j].getPixelXYDouble(jj, (int) (ii + 1.5)) +
m_gList[i][j].getPixelXYDouble(jj, (int) (ii + 0.5)) / 2
- m_gList[i][j].getPixelXYDouble(jj, (int) (ii - 1.5)) +
m_gList[i][j].getPixelXYDouble(jj, (int) (ii - 0.5)) / 2;
double dy = m_gList[i][j].getPixelXYDouble((int) (jj + 1.5), ii) +
m_gList[i][j].getPixelXYDouble((int) (jj + 0.5), ii) / 2
- m_gList[i][j].getPixelXYDouble((int) (jj - 1.5), ii) +
m_gList[i][j].getPixelXYDouble((int) (jj - 0.5), ii) / 2;

                        int iii = ii + 1;
                        int jjj = jj + 1;
                        assert (iii < width && jjj <= height);

                        // Set the magnitude and orientation
imgInterpolatedMagnitude[i][j - 1].setPixelXYDouble(jjj, iii, (int) Math.sqrt(dx
* dx + dy * dy));
imgInterpolatedOrientation[i][j - 1].setPixelXYDouble(jjj, iii, (int)
```

```
((Math.atan2(dy, dx) == M_PI) ? -M_PI : Math.atan2(dy, dx)));

     }
 }

// Pad the edges with zeros
for (int iii = 0; iii < width + 1; iii++) {
imgInterpolatedMagnitude[i][j - 1].setPixelXYDouble(0, iii, 0);
imgInterpolatedMagnitude[i][j - 1].setPixelXYDouble(height, iii, 0);
imgInterpolatedOrientation[i][j - 1].setPixelXYDouble(0, iii, 0);
imgInterpolatedOrientation[i][j - 1].setPixelXYDouble(height, iii, 0);
}

for (int jjj = 0; jjj < height + 1; jjj++) {
     imgInterpolatedMagnitude[i][j - 1].setPixelXYDouble(jjj, 0, 0);
     imgInterpolatedMagnitude[i][j - 1].setPixelXYDouble(jjj, width, 0);
     imgInterpolatedOrientation[i][j - 1].setPixelXYDouble(jjj, 0, 0);
     imgInterpolatedOrientation[i][j - 1].setPixelXYDouble(jjj, width, 0);
    }
  }
}

// Build an Interpolated Gaussian Table of size FEATURE_WINDOW_SIZE
// Lowe suggests sigma should be half the window size
// This is used to construct the "circular gaussian window" to weight
// magnitudes

Matrix G = BuildInterpolatedGaussianTable(FEATURE_WINDOW_SIZE, 0.5 *
FEATURE_WINDOW_SIZE);

// Vector <Double> hist = new Vector<Double>(DESC_NUM_BINS);
double[] hist = new double[DESC_NUM_BINS];

// Loop over all keypoints
for (int ikp = 0; ikp < m_numKeypoints; ikp++) {
     int scale = m_keyPoints.get(ikp).scale;
     float kpxi = m_keyPoints.get(ikp).xi;
     float kpyi = m_keyPoints.get(ikp).yi;
     float descxi = kpxi;
     float descyi = kpyi;
     int ii = (int) (kpxi * 2) / (int) (Math.pow(2.0, (double) scale / Intervals));
     int jj = (int) (kpyi * 2) / (int) (Math.pow(2.0, (double) scale / Intervals));
     int width = m_gList[scale / Intervals][0].xdim;
     int height = m_gList[scale / Intervals][0].ydim;

     Vector<Double> orien = m_keyPoints.get(ikp).orien;
     Vector<Double> mag = m_keyPoints.get(ikp).mag;

/*
 * Find the orientation and magnitude that have the maximum impact
 * on the feature
*/
double main_mag = mag.get(0);
double main_orien = orien.get(0);
for (int orient_count = 1; orient_count < mag.size(); orient_count++) {
     if (mag.get(orient_count) > main_mag) {
      main_orien = orien.get(orient_count);
```

```
        main_mag = mag.get(orient_count);
        }
}

int hfsz = FEATURE_WINDOW_SIZE / 2;
Matrix weight = new Matrix(FEATURE_WINDOW_SIZE, FEATURE_WINDOW_SIZE);
double[] fv = new double[FVSIZE];

for (i = 0; i < FEATURE_WINDOW_SIZE; i++) {
    for (j = 0; j < FEATURE_WINDOW_SIZE; j++) {
            if (ii + i + 1 < hfsz || ii + i + 1 > width + hfsz || jj + j + 1 <
            hfsz || jj + j + 1 > height + hfsz) {
                        weight.set(j, i, 0);
            }
            else {
            weight.set(j, i, G.get(j, i) * imgInterpolatedMagnitude[scale /
            Intervals][scale % Intervals].getPixelXYDouble(jj + j + 1 - hfsz, ii
            + i + 1 - hfsz));
                    }
                }
            }


/*
* Now that we've weighted the required magnitudes, we proceed to
* generating the feature vector
*/
/*
 * The next two two loops are for splitting the 16x16 window into
 * sixteen 4x4 blocks
 */

 for (i = 0; i < FEATURE_WINDOW_SIZE / 4; i++) {
     for (j = 0; j < FEATURE_WINDOW_SIZE / 4; j++) {
          for (int t = 0; t < DESC_NUM_BINS; t++) {
               hist[t] = 0.0;
          }
          // Calculate the coordinates of the 4x4 block
          int starti = (int) ii - (int) hfsz + 1 + (int) (hfsz / 2 * i);
          int startj = (int) jj - (int) hfsz + 1 + (int) (hfsz / 2 * j);
          int limiti = (int) ii + (int) (hfsz / 2) * ((int) (i) - 1);
          int limitj = (int) jj + (int) (hfsz / 2) * ((int) (j) - 1);


         // Go though this 4x4 block and do the thingy
         for (int k = starti; k <= limiti; k++) {
             for (int t = startj; t <= limitj; t++) {
                  if (k < 0 || k > width || t < 0 || t > height) {
                      continue;
                  }

// This is where rotation invariance is done
double sample_orien = imgInterpolatedOrientation[scale / Intervals][scale /
Intervals].getPixelXYDouble(t, k);
sample_orien -= main_orien;
```

```
while (sample_orien < 0) {
       sample_orien += 2 * M_PI;
       }
while (sample_orien > 2 * M_PI) {
       sample_orien -= 2 * M_PI;
      }

assert (sample_orien >= 0 && sample_orien < 2 * M_PI);
       int sample_orien_d = (int) (sample_orien * 180 / M_PI);
assert (sample_orien_d < 360);
       int bin = sample_orien_d / (360 / DESC_NUM_BINS);
    // The bin
double bin_f = (double) sample_orien_d / (double) (360 / DESC_NUM_BINS);
The actual entry
assert (bin < DESC_NUM_BINS);
assert (k + hfsz - 1 - ii < FEATURE_WINDOW_SIZE && t + hfsz - 1 - jj <
FEATURE_WINDOW_SIZE);

hist[bin]+=(1-Math.abs(bin_f-(bin+0.5))) * weight.get(t+hfsz-1-jj,
k+hfsz-1-ii);
       }
     }
     // Keep adding these numbers to the feature vector
     for (int t = 0; t < DESC_NUM_BINS; t++) {
          fv[(i * FEATURE_WINDOW_SIZE / 4 + j) * DESC_NUM_BINS + t] = hist[t];
                    }
                }

      }

/*
 * Now, normalize the feature vector to ensure illumination independence
 */
   double norm = 0;
   for (int t = 0; t < FVSIZE; t++) {
        norm += Math.pow(fv[t], 2.0);
       }
       norm = Math.sqrt(norm);

       for (int t = 0; t < FVSIZE; t++) {
            fv[t] /= norm;
       }

       // Now, threshold the vector
       for (int t = 0; t < FVSIZE; t++) {
           if (fv[t] > FV_THRESHOLD) {
               fv[t] = FV_THRESHOLD;
           }
       }

      // Normalize yet again
      norm = 0;
      for (int t = 0; t < FVSIZE; t++) {
      norm += Math.pow(fv[t], 2.0);
      }
      norm = Math.sqrt(norm);
```

```
            for (int t = 0; t < FVSIZE; t++) {
                fv[t] /= norm;
            }

        }

        // Draw Keypoints
        int[] cscolor = {255, 0, 0};    // points crosses color
        int crosssize = 0;
        Point p1, p2, p3, p4,p5;
        int[] lkcolor = {0, 0, 255};    // links between points color
        int[] draw = {255, 255, 255};
        Image test = src.copyImage(true);
        Image test2 = temp.copyImage(true);


        for (i = 0; i < m_numKeypoints; i++) {

            Keypoint3 kp = m_keyPoints.get(i);

            p1 = new Point((int) kp.xi, (int) kp.yi);
            p2 = new Point((int) kp.xi, (int) kp.yi);

            p3 = new Point((int) kp.xi, (int) kp.yi);
            p4 = new Point((int) (kp.xi + 10 * Math.cos(kp.orien.get(0))), (int)
            (kp.yi + 10 * Math.sin(kp.mag.get(0))));

            drawCross(test, p1,crosssize,cscolor);
            drawCross(test, p2,crosssize,cscolor);
            drawCross(test, p3,crosssize,cscolor);
            drawCross(test, p4,crosssize,cscolor);


        }

        return test;

    }

private static Matrix BuildInterpolatedGaussianTable(int size, double sigma) {
int i, j;
double half_kernel_size = size / 2 - 0.5;

double sog = 0;
Matrix ret = new Matrix(size, size);

assert (size % 2 == 0);

double temp = 0;
for (i = 0; i < size; i++) {
     for (j = 0; j < size; j++) {
          temp = gaussian2D(i - half_kernel_size, j - half_kernel_size, sigma);
               ret.set(j, i, temp);
     }
}
for (i = 0; i < size; i++) {
```

```
    for (j = 0; j < size; j++) {
        ret.set(j, i, 1.0 / sog * ret.get(j, i));
    }
  }
return ret;
}

private static double gaussian2D(double x, double y, double sigma) {
double ret = 1.0 / (2 * M_PI * sigma * sigma) * Math.exp(-(x * x + y * y) / (2.0
* sigma * sigma));
return ret;
}

public static void drawCross(Image image, Point p, int size, int[] color) {
for (int i = p.x - size; i <= p.x + size; i++) {
     image.setVectorPixelXYZTByte(i, p.y, 0, 0, color);

        }
        for (int j = p.y - size; j <= p.y + size; j++) {
            image.setVectorPixelXYZTByte(p.x, j, 0, 0, color);
        }
    }


    /**
     * Method Draw Line
     *
     * @param image
     * @param p1
     * @param p2
     * @param color
     */
    public static void drawLine(Image image, Point p1, Point p2, int[] color) {

        Line link = new Line(p1, p2);
        Point p;
        java.util.Iterator<Point> it = link.iterator();
        while (it.hasNext()) {

            p = it.next();
            image.setVectorPixelXYZTByte(p.x, p.y, 0, 0, color);
        }
    }

    ////////////////////////
    // DEMONSTRATION MAIN //
    ////////////////////////
    public static void main(String[] args) {

        String basepath2 = "F:\\Belajar Java\\database pic\\";
        String imgsrc = "";
        File file = null;
        String choosepath = null;
        String filename = null;


        JFileChooser fc = new JFileChooser(basepath2);
```

```
        int res = fc.showOpenDialog(null);

        if (res == fc.APPROVE_OPTION) {
            file = fc.getSelectedFile();
            imgsrc = file.getPath();
            choosepath = file.getParent();
            filename = file.getName();
        } else {
            JOptionPane.showMessageDialog(null, "You must select one image to be
the reference.", "Aborting...", JOptionPane.WARNING_MESSAGE);
        }


        //Load Image Original
        Image src = ImageLoader.exec(imgsrc);
        Image dump = SIFTrev.exec(src);
        MultiView mv = new MultiView();
        mv = MViewer.exec();
        mv.add(dump);

    }
}

class Keypoint3 {

    public float xi;
    public float yi;    // It's location
    public Vector<Double> mag;  // The list of magnitudes at this point
    public Vector<Double> orien;    // The list of orientations detected
    public int scale;   // The scale where this was detected

    public Keypoint3() {
    }

    public Keypoint3(float x, float y) {
        xi = x;
        yi = y;
    }

public Keypoint3(float x, float y, Vector<Double> m, Vector<Double> o, int s) {
        xi = x;
        yi = y;
        mag = m;
        orien = o;
        scale = s;
    }
}
```