Stage de 2e année de Master Informatique

Recherche par similarité visuelle fine dans les fonds photographiques numérisés

le cnam

par Adrien KRÄHENBÜHL

le 5 septembre 2011

Nancy-Université
Université Henri Poincaré

Remerciements

Je tiens tout d'abord à remercier mes encadrants, Mme Valérie Gouet-Brunet M. Gabriel Bloch, pour m'avoir accompagné durant ce stage et pour le temps qu'ils m'ont consacré.

Je souhaite également remercier Bastien et Thomas, pour leurs précieux conseils et leur bonne humeur. Je remercie aussi l'équipe de Nicéphore Cité, pour son accueil chaleureux et la découverte des spécialités locales.

Je n'oublie pas ma relectrice la plus attentive, Gaëlle, ni les conseils avisés de mon père.

Je remercie enfin tous les membres de ma famille, ainsi que Mathilde et sa maman, qui m'ont soutenu et encouragé dans les moments difficiles.

Table des matières

Introduction

Chalon-sur-Saône, berceau de la photographie où Nicéphore Nièpce réalisa son premier cliché en 1816, la toute première photographie au monde. C'est dans cette ville chargée d'histoire que s'est déroulé le stage concluant deux années de master informatique.

Les photohraphies sont justement au cœur du sujet proposé par Mme Valérie Gouet-Brunet sur la recherche par similarité visuelle dans les fonds photographiques. Le sujet s'inscrit dans un projet de grande envergure visant à proposer aux visiteurs du musée Nicéphore Nièpce une interface innovante pour découvrir de façon ludique le fond photographique du musée. Ils devront être en mesure de parcourir la collection sur des critères visuels tels les couleurs ou les formes.

Le stage va lui se concentrer sur la partie de recherche et d'indexation des images par contenu visuel, qui servira de base à cette interface. Si le sujet de départ annonçait la proposition, le développement et l'évaluation d'une méthode unifiée de description des images du musée, le contenu réel aura été tout autre. Une base logicielle a été créée pour permettre de tester et comparer différentes approches de description locale. Bien que prémice du sujet original, la base logicielle est devenue l'objet majeur du stage par ses contraintes de généricité et de polyvalence. Au terme de sa conception, elle a permis de comparer plusieurs approches de description locale et de les évaluer. Les approches les plus performantes ont été combinées et testées dans l'optique d'une description unifiée des images du musée.

Les approches testées sont des approches de description locale basées sur les points d'intérêt. Elles reposent sur deux outils, souvent étroitement liés : les détecteurs et les descripteurs. Les premiers permettent de localiser des zones intéressantes de l'image et les seconds de les décrire. Forte de cette description des images, la technique des sacs de mots permet d'unifier la représentation des images pour les rendre comparables deux à deux. Des techniques d'indexation viennent enfin proposer des méthodes d'accès rapides pour permettre une utilisation en temps réel inhérente à l'interface utilisateur.

Après une présentation plus en détail des objectifs du stage, nous réaliserons un état de l'art des approches de description locale à base de point d'intérêt. Il sera suivi du détail des différents outils utilisés au sein de la base logicielle. Le chapitre suivant abordera les points techniques de la base logicielle et sa conception à base de modules. Enfin, les résultats des tests effectués seront interprétés avant de présenter les perpectives de poursuite du projet.

CHAPITRE 1

Présentation et objectifs du stage

1.1 Le contexte

Ce stage est le fruit d'une convention de recherche entre trois entités : le Conservatoire National des Arts et Métiers (CNAM), le pôle Nicéphore Cité et le musée Nicéphore Niépce.

C'est au CNAM à Paris qu'est localisée Mme Valérie Gouet-Brunet. Membre de l'équipe Vertigo du laboratoire CEDRIC, elle dirige la partie scientifique du stage.

Le stage s'est déroulé dans les locaux de Nicéphore Cité, pôle de développement, de soutien et d'accompagnement de la filière image et son de la région Bourgogne. Ce pôle localisé à Chalon-sur-Saône, se compose de 14 personnes, dont le directeur adjoint, M. Gabriel Bloch, personne référente pour ce stage. Fort de nombreux partenariats, Nicéphore Cité s'investit dans divers projets innovants pour mener à bien ses missions :

- La formations aux médias audiovisuels
- La recherche et le développement pemettant d'accroître la compétitivité des entreprises
- L'hébergement et l'accompagnement de jeunes entreprises en développement
- Des prestations autour de l'image et du son
- La sensibilisation et l'éveil aux nouvelles technologies

Le musée Nicéphore Niépce, musée de la photographie, est lui situé à deux pas de Nicéphore Cité. Il possède une collection de plus de trois millions de photographies retraçant l'histoire de la photographie, de son invention à Chalon-sur-Saône par Nicéphore Niépce aux photographies d'aujourd'hui. L'intégralité de la collection est en cours de numérisation depuis huit ans et trois-cent-mille photographies sont aujourd'hui numérisées. C'est sur un échantillon de ces photographies et avec l'aide du responsable du service inventaire-documentation, Sylvain Besson, qu'ont été établies les bases du sujet de stage.

FIGURE 1.1 – Exemple de photographies de la collection du musée. Au dessus figure deux unes de revues et les photographies originales de Erwin Blumenfeld. En bas à gauche, une photographie de André Kertész et la page de la revue « Art et Medecine » de novembre 1932. À gauche, un reportage d'André Steiner pour la revue « Marianne ».

1.2 Le sujet

Le sujet de stage, dont l'énoncé se trouve en annexe B page 81, peut se résumer ainsi :

> *Étudier et évaluer les différentes méthodes de descriptions locales de contenu visuel par points d'intérêt et proposer une approche unifiée et paramétrable de description et d'indexation des images du musée Nicéphore Niépce.*

Définition : Un **point d'intérêt** dans une image est une zone de pixels aux propriétés remarquables. Il s'agit le plus souvent d'une zone comportant des changements soudains d'intensité. Dans les approches les plus récentes, il peut également s'agir d'une vaste zone sans changements d'intensité ou encore d'une zone comportant une ou plusieurs symétries locales. Ces zones peuvent prendre la forme de points, de courbes, ou encore de régions rectangulaires, circulaires, elliptiques, etc.

La technique servant de trame au stage et à la base logicielle est la technique des sacs de mots. Il s'agit d'une technique couramment utilisée en reconnaissance d'images par description locale. Elle permet d'indexer efficacement un ensemble d'images à partir de la description de leur contenu.

Présentée dans l'article [SZ03] où elle est appliquée aux points d'intérêt, son principe est le suivant :

1. Pour chaque image, détecter les points d'intérêt et en calculer un vecteur descripteur.
2. Prendre l'ensemble des vecteurs descripteurs obtenus et en réaliser une classification, algorithme qui permet leurs trouver un nombre restreint de représentants.
3. Prendre l'ensemble des représentant pour constituer le sac de mots, aussi appelé vocabulaire.
4. Pour chaque vecteur descripteur de chaque image, lui attribuer le mot le plus proche.
5. Décrire chaque image par un vecteur représentant la proportion de chaque mot qu'elle contient. Ce vecteur est appelé signature de l'image. Sa taille est égale à la taille du vocabulaire, c'est à dire le nombre de mots du sac de mots.

Cette technique doit constituer le socle d'une application permettant de retrouver des images similaires au sein de la collection du musée Nicéphore Niépce. Les attentes du musée concernent la mise en correspondance de photographies originales publiées dans des revues. Plusieurs scénarios de recherche peuvent être imaginés :

- retrouver la revue dans laquelle a été publiée une photographie originale,
- retrouver toutes les photographies d'une séance de prise de vues ayant amené à la publication d'une photographie dans une revue,
- retrouver l'image originale d'une photographie remake

Cependant, les images publiées ont pu subir des changements de différentes natures avant publication : modification des couleurs, recadrage, déformations... Les algorithmes actuels ne permettent pas toujours de reconnaitre tous ces scénarios et ces différences. L'idée est donc de proposer une méthode paramétrable permettant de retrouver les images similaires suivant un scénario ou une transformation donnée.

Ce vaste sujet de recherche ne s'est pas conclu par la proposition d'une nouvelle approche d'indexation unifiée, mais par la création d'une base logicielle permettant de les expérimenter. Il s'agissait en début de stage d'une première étape qui s'est finalement étendue à l'intégralité du stage. Cette base doit être la plus modulable possible afin de permettre par la suite la proposition d'approches plus sophistiquées. Comme nous le verrons par la suite, plusieurs fonctionnalités sont venues enrichir la base logicielle au fur et à mesure du stage.

CHAPITRE 2

État de l'art

Les techniques de descriptions locales de contenu visuel à base de points d'intérêt reposent sur deux outils qui seront définis plus loin : les détecteurs et les descripteurs. L'utilisation combinée de ces deux outils complémentaires a pour objectif d'obtenir une description robuste des images. Une description robuste est une description permettant d'identifier des points d'intérêt identiques dans des images décrivants une même scène ou un même objet ayant subi diverses transformations. On dit alors que les détecteurs et les descripteurs possèdent des propriétés d'invariance à ces transformations.

FIGURE 2.1 – Exemple de détection de points d'intérêt invariante aux rotations.
Source : `http://www.developpez.net/`.

Les transformations peuvent être de différentes natures : géométriques comme sur la figure 2.1, mais aussi photométriques, colorimétriques ou encore au bruit. Nous verrons par exemple des détecteurs et des descripteurs invariants aux rotations. Pour les détecteurs, cela signifiera qu'ils sauront obtenir les mêmes points d'intérêt dans une image et la même image ayant subi une rotation, comme l'illustre la figure 2.1. Pour les descripteurs, cela signifiera que pour un même point d'intérêt identifié dans ces deux images, le vecteur descripteur obtenu sera très similaire.

La figure 2.1 donne un exemple de détection invariante aux rotations. On peut noter que certains points comme celui à la commissure des lèvres ne sont détectés que dans une seule image. Cela est dû à l'apparition de bruit à proximité de la bouche.

Ce chapitre présente tout d'abord les principaux détecteurs et descripteurs que l'on retrouve dans la littérature, notamment dans les articles [TM08], [LA08] et [MTS+05].

Plusieurs autres outils vont ensuite être expliqués. Ils constituent le socle de l'outil de recherche d'images similaires qui sera présenté au chapitre 3 page 30. Ces outils sont les sac de mots, l'indexation inversée, la recherche des plus proches voisins, la recherche d'images similaires et la combinaison de détecteurs/descripteurs.

2.1 Les détecteurs

Les détecteurs permettent d'isoler des zones d'intérêt dans une image. Ces zones d'intérêt peuvent être de deux types : des coins ou des régions.

Le terme de coin désigne un changement soudain d'une caractéristique au voisinage d'un pixel. Il peut donc s'agir d'un coin au sens commun du terme, mais également d'un point isolé, d'un bord à forte courbure, d'un changement de couleurs, etc.

Le terme de région désigne quant à lui une zone de pixels contiguës qui satisfait une certaine propriété. Cette propriété peut être par exemple exprimée en terme d'homogénéité, de couleur, de saturation, etc.

Voici maintenant le détail de certains détecteurs importants, qui ont été incorporés à la base logicielle ou testés à partir de codes publics existants.

2.1.1 Les détecteurs de coins

2.1.1.1 Moravec

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations d'angles $k \times 45^\circ, k \in \mathbb{N}$	non	non	non

Référence : [Mor77]

Principe : Ce détecteur détermine le changement d'intensité moyen dans un voisinage donné.

Détails : On choisit le voisinage d'un pixel comme une fenêtre (F1) entourant ce pixel. On décale légèrement (F1) dans une direction donnée pour obtenir une nouvelle fenêtre (F2), comme sur la figure 2.2. La somme des carrés de la différence des intensités des pixels aux mêmes positions dans (F1) et (F2) est notée S. Moravec calcule S est pour un décalage dans chacune des directions horizontales et verticales. Les points d'intérêt sont déterminés ainsi :

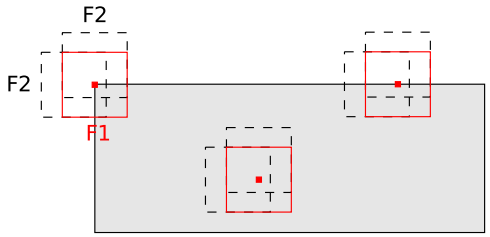


FIGURE 2.2 – Les trois situations possibles pour le détecteur de Moravec

- Si S est petite pour les deux directions, on se trouve dans une zone d'intensité constante.
- Si S est petite pour une direction et grande pour l'autre, on se trouve sur un bord.
- Si S est grande dans les deux directions, on se trouve sur un coin ou un point isolé.

2.1.1.2 Harris

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations	non	oui	non

Référence : [HS88] et [Sch96]

Principe : Ce détecteur améliore celui de Moravec en apportant de nouvelles invariances.

Détails : Harris apporte les améliorations suivantes :

Correction de l'anisotropie Utilisation des dérivées en x et en y de (F1) au lieu de la seconde fenêtre (F2) que l'on voit sur la figure 2.2 page précédente. Cette dérivée peut être calculée par exemple avec un filtre de Prewitt et Sobel. La correction de l'anisotropie permet une invariance aux rotations.

Correction de la réponse forte des bords Utilisation de la matrice de Harris, notée M. Les valeurs propres de M correspondent aux courbures principales de la fonction d'autocorrélation et ont un comportement identique au S du détecteur de Moravec.

Le détecteur R que propose Harris est le suivant :

$$\mathrm{R}(x,y) = \mathrm{D}et(\mathrm{M}(x,y)) - k.\mathrm{T}r(\mathrm{M}(x,y))^2$$

k est un coefficient que l'on peut ajuster. Dans ses expérimentations, Harris a obtenu les meilleurs résulatts avec $k = 0,04$. Il se comporte ainsi :

- Si R est petit : on se trouve dans une zone d'intensité constante.
- Si R est grand négatif : on se trouve sur un bord.
- Si S est grand positif : on se trouve sur un coin ou un point isolé.

En 1996, Schmid apporte une nouvelle amélioration pour une meilleure invaraince au bruit. Elle utilise un filtre Gaussien pour remplacer le masque binaire. Cela revient à ajouter une pondération aux termes de la somme de Moravec. Cette étape est réalisée préalablement à la correction de l'anisotropie car c'est le calcul des dérivées qui est sensible au bruit. Cette nouvelle version du détecteur de Harris porte le nom de Harris précis.

2.1.1.3 Harris couleur

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations	oui	oui	oui

Référence : [MGD98]

Principe : Ce détecteur améliore le détecteur Harris précis en le rendant invariant aux changements colorimétriques.

Détails : L'image est analysée sous ses trois composantes couleurs. Une image couleur I est décomposée en trois images R, V et B pour chacune des composantes vertes, bleues et rouges. La matrice M de Harris devient la suivante :

$$\mathrm{M}_\sigma = \begin{pmatrix} \mathrm{G}_\sigma(\mathrm{R}_x^2 + \mathrm{V}_x^2 + \mathrm{B}_x^2) & \mathrm{G}_\sigma(\mathrm{R}_x\mathrm{R}_y + \mathrm{V}_x\mathrm{V}_y + \mathrm{B}_x\mathrm{B}_y) \\ \mathrm{G}_\sigma(\mathrm{R}_x\mathrm{R}_y + \mathrm{V}_x\mathrm{V}_y + \mathrm{B}_x\mathrm{B}_y) & \mathrm{G}_\sigma(\mathrm{R}_y^2 + \mathrm{V}_y^2 + \mathrm{B}_y^2) \end{pmatrix}$$

où G_σ est une gaussienne de paramètre σ.

2.1.1.4 Harris-Laplace

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	non	oui	non

Référence : [MS04]

Principe : Il combine le détecteur de coins de Harris et la fonction de Laplace pour devenir invariant aux changements d'échelles.

Détails : Cet algorithme commence par simuler les changements d'échelle par produit de convolution entre l'image initiale et des gaussiennes de paramètres σ croissants. Un exemple de simulation est donné par la figure 2.3. Puis le détecteur de Harris est appliqué à chaque

(a) echelle initiale (b) $\sigma = \sqrt{2}$ (c) $\sigma = 2$ (d) $\sigma = 2\sqrt{2}$ (e) $\sigma = 4$

FIGURE 2.3 – Exemple de simulations de changements d'échelles par des gaussiennes de paramètre σ croissant. Il y a un facteur $\sqrt{2}$ entre deux échelles successives.

échelle pour trouver les extremums locaux. Ensuite intervient la fonction de Laplace : pour chaque extremum trouvé, on cherche à déterminer s'il est aussi un extremum lors d'un changement d'échelle. Pour cela on calcule le LoG (Laplacian of Gaussians) à l'échelle supérieure et à l'échelle inférieure. L'extremum est un point d'intérêt s'il est aussi extremum des deux échelles voisines.

2.1.1.5 Hesse-Laplace

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	non	oui	non

Référence : [MS04]

Principe : Accèlere le détecteur Harris-Laplace en modifiant la matrice du calcul de R.

Détails : Le protocole est le même que pour Harris-Laplace. L'unique différence intervient au niveau du détecteur de Harris : ce n'est plus sur la matrice de Harris M mais sur la matrice de Hesse H qu'est calculé R. La formule reste la même :

$$\mathrm{R}(x,y) = \mathrm{D}et(\mathrm{H}(x,y)) - k.\mathrm{T}r(\mathrm{H}(x,y))^2$$

avec $k = 0,04$

2.1.1.6 SIFT (Scale Invariant Feature Transform)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	partiellement	oui	non

Référence : [Low99] et [Low04]

Principe : Cette méthode combine un détecteur avec un descripteur. Elle cherche les maximums locaux sur une série de différences d'images gaussiennes.

Détails : La partie détection de SIFT est basée sur une DoG (Difference of Gaussians) :

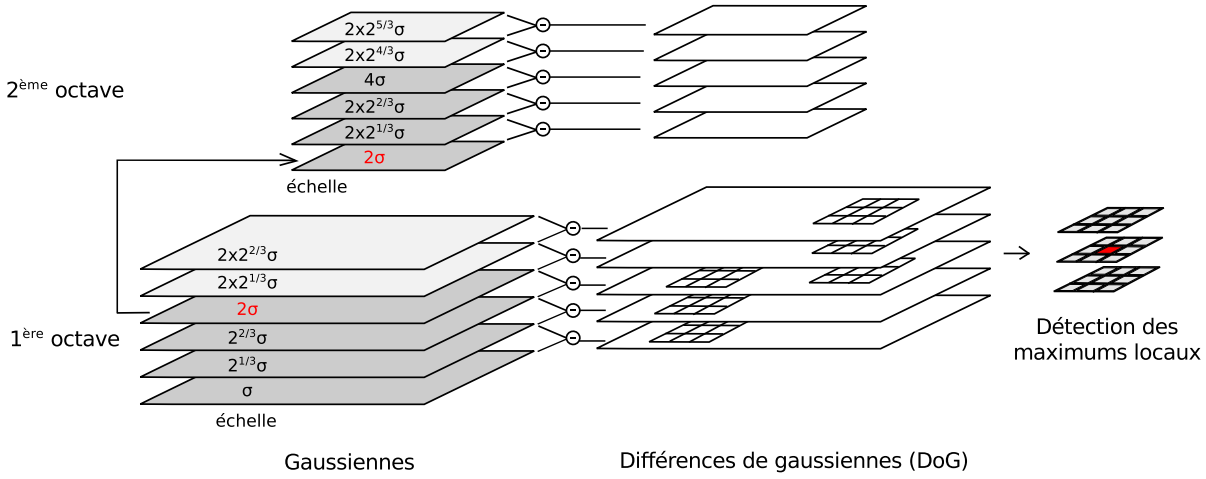


FIGURE 2.4 – Illustration de la détetcion de l'algorithme SIFT.

Construction des octaves Le détecteur de la méthode SIFT commence par la construction des *octaves*. Une octave est un ensemble d'images d'échelles croissantes. L'échelle est doublée entre la première image d'une octave et la dernière. Le nombre d'échelles n total d'une octave peut varier. Le coefficient multiplicatif entre deux échelles consécutives est donc $2^{1/(n-1)}$ afin de répartir les échelles uniformément. Sur la figure 2.4 et dans l'article de Lowe, n vaut 4. Le facteur multiplicatif est donc $2^{1/3}$. On commence une nouvelle octave avec la dernière échelle de l'octave précédente. La première octave commence par l'image originale dont on considère que son échelle est σ.

On voit sur la figure 2.4 que deux autres échelles ont été ajoutées à la fin de chaque octave. Ce sont en fait les deuxièmes et troisièmes échelles de l'octave supérieure. Elles ne sont calculées qu'une seule fois mais Lowe les ajoutent à l'octave précédente pour que les étapes suivantes soient réalisées indépendamment sur chaque octave.

Construction des DoG Sur chacune des octaves, on calcule les DoG de deux images de σ successifs. On obtient donc $n+1$ DoG pour chaque octave, du fait des deux échelles supplémentaires introduites par Lowe.

Détection des extremums Les extremums sont déterminés à chaque octave sur chaque groupe de trois DoG successives (1-2-3, 2-3-4 et 3-4-5). Les extremums sont les pixels qui ont l'intensité la plus basse ou la plus haute de leur voisinage. Ce voisinage comporte 26 pixels : les 8 appartenant au masque 3x3 centré sur le pixel et les 9 des masques 3x3 des images supérieures et inférieures. Au final, un extremum est donc identifié par ses coordonnées dans l'image de base et l'échelle à laquelle il a été identifié. Les coordonnées des points d'intérêt des octaves supérieures sont ramenées à l'échelle initiale par interpolation.

Identification des points d'intérêt Tous les extremums ne sont pas pertinents. Cette étape propose donc de supprimer d'abord les extremums de faible contraste par seuilllage. Sont ensuite évincés les extremums situés sur des arêtes, en se basant sur les rapports des directions principales (la trace et le déterminant de la matrice Hessienne). En effet, un bord aura un grand ratio dans une seule direction.

La construction des vecteurs descripteurs à partir de ces extremums est traitée dans la section 2.2.2 page 19.

2.1.1.7 SURF (Speeded-Up Robust Features)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et echelles	oui	oui	non

Référence : [BETVG08]

Principe : Cette méthode s'inspire de SIFT et utilise des approximations pour augmenter la rapidité des traitements.

Détails : SURF se base sur la matrice de Hesse, résultat du produit de convolution d'une gaussienne avec la matrice des dérivées secondes. Cette matrice 2×2 est approximée par la matrice des dérivées partielles d'ordre deux, ce qui revient à effectuer un calcul de surface pour chacun des quatres termes. Le calcul de ces surfaces est accéléré par la construction préalable de l'image intégrale. Pour le changement d'échelle, ce n'est plus l'image qui est redimensionnée comme pour SIFT, mais le masque sur lequel est calculée la matrice hessienne. Sont donc redéfinies les notions d'octave et d'échelle :

Octave Une octave est maintenant un ensemble de 4 masques de tailles croissantes incrémentées linéairement. Le premier masque de la première octave a une taille de 9×9 est est incrémenté de 6 en 6. Chaque nouvelle octave commence par le second masque de l'octave précédente.

Échelle Une échelle est déterminée par la taille du masque. L'échelle 1 correspond au masque de plus petite taille.

Une fois les échelles calculées et les octaves déterminées, la localisation des points d'intérêt se fait comme avec SIFT : ce sont les pixels qui sont extremum de leur voisinage (de taille $3 \times 3 \times 3$). Les coordonnées des points d'intérêt sont ensuite ramenées dans le repère initial par interpolation. Chaque point d'intérêt est donc maintenant identifié par sa position et son échelle. La construction des vecteurs descripteurs est expliquée dans la section 2.2.3 page 20.

2.1.2 Les détecteurs de régions

2.1.2.1 Les détecteurs Harris-affine et Hesse-affine

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	transformations affines	non	oui	non

Référence : [MS04]

Principe : Amélioration des deux détecteurs Harris-Laplace et Hesse-Laplace pour les rendre invariants à toutes transformation affine.

Détails : Ce détecteur commence par déterminer les points d'intérêt avec le détecteur de Harris-Laplace ou Hesse-Laplace. Il se déroule ensuite en quatre étapes qui peuvent être itérées pour chaque point détecté :

1. On estime sa forme affine par une ellipse grâce à la matrice des moments d'ordre deux.
2. On normalise la région elliptique pour obtenir un cercle.
3. On détecte la nouvelle position et la nouvelle échelle de l'image normalisée.

4. On recommmence à la première étape si les valeurs propres de la matrice des moments d'ordre deux de l'image normalisée sont différentes de celles de la matrice de la première étape.

2.1.2.2 Le détecteur EBR (Edge-Based Region)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	non	non	non

Référence : [TVG04]

Principe : Combine le détecteur de coins de Harris avec le détecteur de bords de Canny pour construire des régions d'invariance sous forme de parallélogrammes.

Détails : Les deux premières étapes sont indépendantes. Il s'agit de déterminer les points d'intérêt de Harris et les contours de Canny. Puis pour chaque point d'intérêt p, on détecte le contour le plus proche, qui sera généralement très proche. On fait alors bouger deux points le long de ce contour, en partant du point du contour le plus proche de p et en partant dans les deux directions opposées. Sont ainsi obtenus plusieurs parallélogrammes successifs. On retient le parallélogramme dont la surface représente un extremum d'une fonction d'intensité.

2.1.2.3 Le détecteur IBR (Intensity-Based Region)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	non	non	non

Référence : [TVG04]

Principe : Cet algorithme construit des régions elliptiques en se basant sur l'intensité de l'image.

Détails : La première étape consiste à identifier les maximums locaux en intensité. Pour chaque maximum p identifié, on observe les valeurs d'une fonction dépendant de l'intensité le long des rayons issus de p. Dès que la valeur de cette fonction change de signe, c'est un extremum. On relie les extremums des rayons voisins de façon à définir une région autour de p. Enfin, on approxime ces régions par des ellipses : ce sont les régions d'intérêt.

2.1.2.4 Le détecteur MSER (Maximally Stable Extremal Regions)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	transformations affines	non	non	non

Référence : [MCUP02]

Principe : Méthode de détection de zones d'intérêt basée sur le principe de ligne de partage des eaux.

Détails : Le principe de base est le seuillage incrémental de l'intensité en niveaux de gris de l'image globale. Le seuil part de 0 et est incrémenté progressivement. Une MSER est une région qui reste stable dans un large intervalle de seuils. Les bords des MSER ont donc une bordure à fort contraste avec les premiers pixels extérieurs à la région.

2.1.2.5 Le détecteur MSCR (Maximally Stable Colour Regions)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	transformations affines	non	non	oui

Référence : [For07]

Principe : Extension du détecteur MSER aux images en couleurs.

Détails : Basé sur le même principe de seuillage que MSER, la différence se situe au niveau de la définition d'une MSER. Une MSER est ici un ensemble de pixels de couleurs proches et similaires. De plus, Per-Erik Forss'en apporte également deux améliorations. La première est une nouvelle façon de détecter les bords, basée sur la loi de Poisson. La seconde concerne la détection des régions et améliore l'invariance au bruit et aux changements d'échelles.

2.1.2.6 Le détecteur de régions saillantes

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations, échelle	non	non	non

Référence : [KZB04]

Principe : Les régions sont localisées en se basant sur une fonction d'entropie.

Détails : On commence par calculer pour chaque pixel l'entropie de la fonction de densité de probabilité pour une région elliptique centrée sur ce pixel. On cherche ensuite les extremums locaux de cette entropie. Puis on calcule la dérivée de la densité de probabilité pour ces extremums. La saillance d'une région est donnée par le produit de la fonction d'entropie avec cette dérivée. Finalement, on ne conserve comme région d'intérêt que les régions dont la saillance est supérieure à un seuil donné.

2.1.2.7 Le détecteur de symétries locales

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	partiellement	non	oui

Référence : [Hei04]

Principe : Localise des points d'intérêt comme des centres de symétries locales.

Détails : Ce détecteur propose de détecter des points au centre de symétries locales. Il se base sur la détection des bords. La symétrie intervient entre les orientations des gradients couleur des pixels constituant les bords détectés. Sont détectées les symétries centrales et axiales.

Pour résumer, ils considère deux points p_i et p_j reliés par une droite de centre c et calcule deux coefficients compris entre 0 et 1 :

- $\mathrm{PWF}^{-}_{col}(p_i, p_j)$ détermine si les gradients en p_i et p_j sont parrallèles. Il vaut 1 si les deux gradients sont parallèles et 0 s'ils sont perpendiculaires.
- $\mathrm{PWF}^{+}_{col}(p_i, p_j)$ détermine si les gradients en p_i et p_j sont en symetrie de centre c. Il vaut 1 si les deux gradients sont symetriques et 0 s'ils ne le sont pas du tout.

Le détecteur final PWF_{col} vaut :

$$\mathrm{PWF}_{col}(p_i, p_j) = \mathrm{PWF}^{-}_{col}(p_i, p_j) \cdot \mathrm{PWF}^{+}_{col}(p_i, p_j)$$

On applique ensuite un seuil à PWF_{col}. Tous les points d'intérêt dont le coefficient PWF_{col} est supérieur à ce seuil sont conservés.

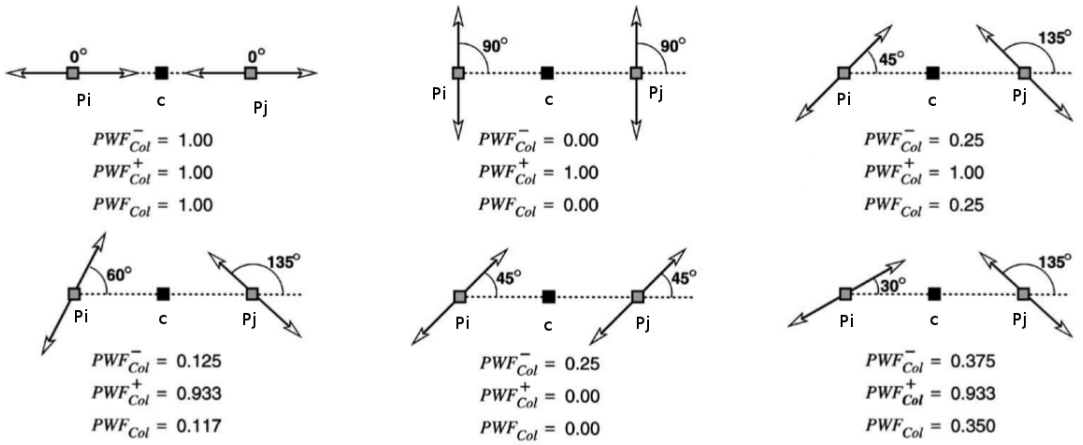


FIGURE 2.5 – Exemple de coéfficients PWF^{-}_{col}, PWF^{+}_{col} et PWF_{col} pour différents cas de figures.
Source : figure extraite de l'article Hei04.

2.2 Les descripteurs

Les descripteurs ont pour but de décrire les points d'intérêt obtenus par un détecteur. Ils analysent le voisinage proche de chaque point, le support d'extraction, pour produire un vecteur caractéristique de cette zone. Ce vecteur, appelé également descripteur dans la littérature, sera appelé ici vecteur descripteur pour plus de clarté. L'objectif d'un descripteur est d'obtenir des vecteurs descripteurs uniques pour chaque point d'intérêt.

Deux aspects sont prépondérants et indissociables pour un descripteur : l'invariance et la discriminance. L'invariance correspond à la capacité à décrire un même point d'intérêt de la même manière dans deux images distinctes. Un détecteur peut ainsi être invariant aux transformations géométriques, photométriques, au bruit, etc. La discriminance est la capacité à discerner deux points d'intérêt différents dans deux images distinctes. Ces deux aspects sont en conflit constant : plus un descripteur sera invariant, moins il sera discriminant, et inversement. L'invariance d'un point d'intérêt s'obtient au prix d'une réduction de l'information qu'il contient. Tout descripteur doit donc faire un compromis entre invariance et disciminance.

Les descripteurs décrits ci-après vont être présentés avec leurs invariances, mais il est important de ne pas oublier le critère de discriminance.

2.2.1 Les filtres orientables

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations	non	non	non

Référence : [FA91] et [SF96]

Principe : Combinaison linéaire de filtres de base pour déterminer l'orientation d'un point d'intérêt.

Détails : Les filtres orientables ont été intoduits par Freeman et Adelson en 1991. L'idée est de combiner deux filtres de base obtenus à partir de la dérivée partielle d'ordre 1 d'une gaussienne. Ce sont les deux filtres :

$$\mathrm{G}_1^0(x,y) = \frac{\partial}{\partial x} e^{-(x^2+y^2)} \quad et \quad \mathrm{G}_1^{\frac{\pi}{2}}(x,y) = \frac{\partial}{\partial x} e^{-(x^2+y^2}$$

où l'indice donne l'ordre de la dérivée et l'exposant sa direction. Ces deux filtres calculent respectivement les réponses aux orientations de 0°et 90°pour les points de coordonnées (x, y). En combinant les deux filtres, on peut obtenir les réponses de n'importe quelle orientation avec :

$$G_1^{\theta}(x, y) = \cos(\theta) \cdot G_1^{0}(x, y) + \sin(\theta) \cdot G_1^{\frac{\pi}{2}}(x, y)$$

En 1996, Simoncelli et Farid appliquent les filtres orientables aux points d'intérêt.

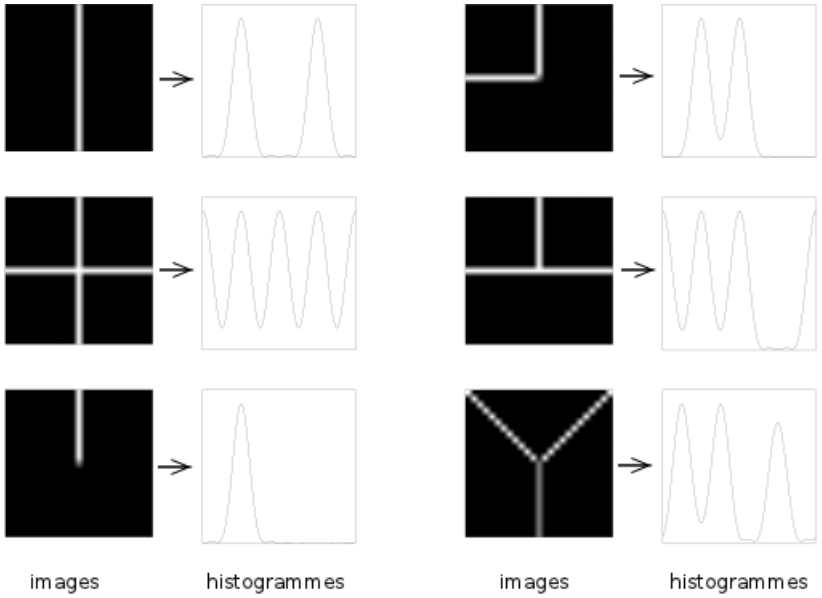


FIGURE 2.6 – Mise en correspondance d'images et de leurs histogrammes d'orientations obtenus avec 18 filtre orientables. Source : figure extraite de l'article SF96.

Ils commencent par déterminer un ensemble de filtres d'orientations croissantes dans l'intervalle $[0, 2\pi]$. Pour chaque point d'intérêt, ils construisent l'histogramme de ses orientations à partir des réponses de chaque filtre. Des exemples d'histogrammes sont donnés par la figure 2.6, à droite des supports d'exraction des point d'intérêt.

Simoncelli et Farid proposent également de nouveaux filtres de base à partir de dérivées d'ordres supérieurs.

2.2.2 SIFT (Scale Invariant Feature Transform)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	partiellement	oui	non

Référence : [Low99] et [Low04]

Principe : Cette méthode combine un détecteur avec un descripteur. Elle cherche les maximum locaux sur une série de différences d'images gaussiennes.

Détails : Cette partie fait suite à la détection des points d'intérêt par la méthode SIFT de la section 2.1.1.6 page 13. Nous supposons donc les points d'intérêt identifiés, avec leurs coordonnées dans le repère de l'image référence et leur échelle d'extraction.

Pour chaque point d'intérêt $p(x, y, \sigma)$, il faut réaliser les étapes suivantes. On considèrera un voisinage de 16×16 pixels autour de $p(x, y, \sigma)$:

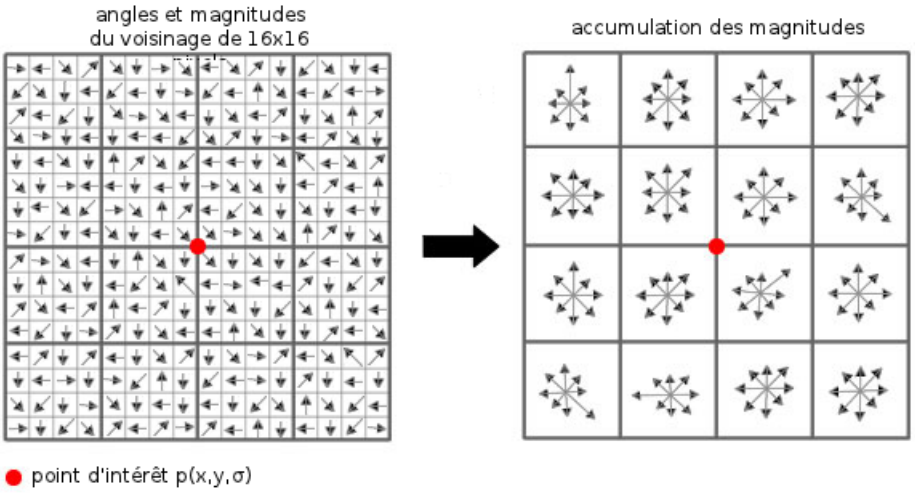


FIGURE 2.7 – Illustration des étapes de description des points d'intérêt de SIFT.

Calcul de l'angle et de la magitude Pour chacun des 256 pixels du voisinage, on calcule sa magnitude $m(x,y,\sigma)$ et son angle $\theta(x,y,\sigma)$ à partir du LoG de l'image, comme le montre la figure 2.7 :

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

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

Le LoG de l'image permet de donner plus de poids aux voisins proches.

Accumulation des magnitudes On divise la fenêtre de voisinage en 16 fenêtre de 4×4 pixels. Pour chacune des fenêtres, on réalise un histogramme d'accumulation des magnitudes en fonction de l'angle. L'accumulation est représentée à droite sur la figure 2.7. Cet histogramme comporte 8 classes d'angles d'amplitude 45°($[-22,5;22,5[, [22,5,67,5[$, etc.) Pour un pixel dont l'orientation du gradient est de 10°, on ajoutera donc la valeur de sa magnitude à la classe $[-22,5;22,5[$.

Création du vecteur descripteur Le vecteur descripteur contient au final un angle, une échelle et le vecteur de taille 128 résultant de la concaténation des histogrammes.

2.2.3 SURF (Speeded-Up Robust Features)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et echelles	oui	oui	non

Référence : [BETVG08]

Principe : Cette méthode s'inspire de SIFT et utilise des approximations pour augmenter la rapidité des traitements.

Détails : Cette partie fait suite à la détection des points d'intérêt traitée à la section 2.1.1.7 page 15.

Calcul de l'orientation Cette première étape repose sur les ondelettes de Haar. Elles sont calculées au voisinage de chaque point d'intérêt selon x et selon y, ce voisinage étant un cercle de rayon 6x l'échelle. Les réponses de ces ondelettes sont représentées dans un

espace à deux dimensions centré sur le point d'intérêt, avec la force des ondelettes selon x sur l'axe des abscisses et la force selon y sur l'axe des ordonnées. On fait ensuite tourner une fenêtre centrée sur le point d'intérêt et d'angle $\frac{\pi}{3}$. Pour chaque position successive de cette fenêtre, on somme les forces des réponses d'ondelettes qu'elle contient. L'orientation finale du point d'intérêt est la direction moyenne de la fenêtre qui obtient la somme maximale. Notons que le calcul des ondelettes peut être accéléré par le calcul de l'image intégrale.

Construction du vecteur descripteur Il est calculé sur un voisinage carré de taille 20x l'échelle et orienté selon l'angle obtenu à l'étape précédente. Ce voisinage est découpé en 16 régions et chaque région en 4 sous-régions. Dans chacune des sous-région est calculé un vecteur (d_x, d_y). Là encore il est obtenu à partir des ondelettes de Haar. Son abscisse (resp. son ordonnée) est la somme des réponses selon x (resp. y). On construit alors le vecteur v de chaque région à partir des 4 vecteurs de ses sous-régions :

$$v = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|)$$

Le vecteur descripteur est la concaténation des 16 vecteurs v. Il est donc de dimension 64.

Une extension possible de SURF est SURF-128. Cette extension sépare chaque terme du vecteur v en 2 parties pour donner le vecteur v_{128} :

$$v_{128} = (\sum_{d_y<0} d_x, \sum_{d_y\geq 0} d_x, \sum_{d_x<0} d_y, \sum_{d_x\geq 0} d_y, \sum_{d_y<0} |d_x|, \sum_{d_y\geq 0} |d_x|, \sum_{d_x<0} |d_y|, \sum_{d_x\geq 0} |d_y|)$$

Ce nouveau vecteur de dimension est donc de dimension 128 et donne de meilleurs résultats tout en étant aussi rapide à calculer.

2.2.4 ASIFT (Affine-SIFT)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	transformations affines	partiellement	oui	non

Référence : [MY09]

Principe : Cette méthode utilise SIFT sur plusieurs images qui simule un changement de point de vue. Cela permet de devenir invariant à toutes les transformations affines au prix d'un temps de calcul bien plus long.

Détails : A l'instar de SIFT, ASIFT couple un détecteur et un descripteur. A partir de l'image de base, on commence par calculer de nouvelles images en simulant des changements de point de vue. SIFT est ensuite appliqué sur l'ensemble des images produites et sur l'image de base. L'ensemble des points obtenus est réduit par une méthode d'estimation de paramètres, par exemple RANSAC (RANdom SAmple Consensus).

2.2.5 CSIFT (Color-SIFT)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	partiellement	oui	oui

Référence : [AHF06]

Principe : Adaptation de SIFT à la couleur.

Détails : Il s'agit d'appliquer SIFT dans les trois composantes de couleurs : rouge, vert et bleu. On obtient ainsi un vecteur descripteur 3 fois plus grand que celui de SIFT. Une amélioration possible est de ne conserver que le vecteur le plus discriminant. Dans l'article référence, il ne s'agit plus de traiter séparement les trois couleurs mais de prendre réellement en compte les caractéristiques géométriques et colorimetriques. Les caractéristiques colorimétriques sont obtenues par étude de la réflectance.

2.2.6 GLOH (Gradient Location and Orientation Histogram)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	oui	oui	non

Référence : [MS05]

Principe : Amélioration de SIFT qui optimise les régions des histogrammes.

Détails : Le processus est identique point pour point à celui de SIFT. La première différence se situe au niveau du voisinage considéré. Pour SIFT, il s'agit d'un voisinage rectangulaire découpé en 4 zones. Pour GLOH, le voisinage est circulaire. Il est constitué de 17 zones, réparties en une zone centrale entourée de deux anneaux découpés chacun en 8 zones. La seconde différence est alors le nombre d'orientations considérées dans chaque zone. Il y avait 8 zones pour SIFT et il y en a 16 pour GLOH. On obtient donc des vecteurs de dimension 272, dimension ensuite réduite par ACP (Analyse en Composantes Principales) pour arriver à 64 dimensions.

2.2.7 HOG (Histogram of Oriented Gradients)

Type d'invariance	Géométrique	Photométrique	Au bruit	Colorimétrique
Est invariant	rotations et échelles	oui	oui	oui

Référence : [DT05]

Principe : Utilisation d'une grille dense pour calculer des histogrammes locaux d'orientation.

Détails : Ce descripteur se base sur un découpage en grille de l'image initiale.

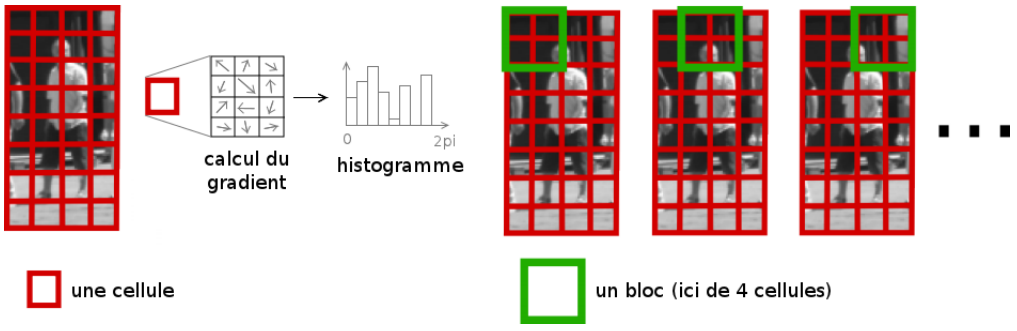


FIGURE 2.8 – Légende

L'algorithme, illustré par la figure 2.8, se compose des étapes suivantes :

Calcul du gradient Le gradient est calculé en chaque pixel de l'image par les filtres dérivatifs $[-1,0,1]$ et $[-1,0,1]^T$. Des filtres plus complexes comme les filtres de Prewitt et Sobel dégradent les performances.

Calcul des histogrammes L'image est donc découpée en cellules de petites tailles, la taille la plus performante étant de 6 × 6 d'après l'article. L'histogramme est construit à partir des orientations en chaque pixel de la cellule, pondérées par l'intensité du gradient. Encore d'après l'article, le nombre d'orientations optimal de l'histogramme est de 9. Nous obtenons au terme de cette étape un histogramme par cellule.

Normalisation des histogrammes Pour normaliser les histogrammes, les cellules sont regroupées en blocs. Chaque bloc est ensuite normalisé indépendamment des autres. Les blocs pouvant se chevaucher, une cellule peut participer plusieurs fois au vecteur descripteur final de l'image.

Les vecteurs descripteurs sont finalement construits à partir des valeurs des histogrammes normalisés.

L'ensemble de ces détecteurs et descripteurs couvre donc la plupart des invariances considérées, chacun ayant son invariance privilégiée. Lorsqu'un détecteur est combiné à un descripteur, les vecteurs descripteurs qui en résultent sont aussi appelés des caractéristiques locales. Celles-ci sont propres à un couple détecteur/descripteur et une image peut donc avoir autant de descriptions que de combinaisons compatibles. Il est de ce fait intéressant de chercher des combinaisons qui offrent de bons compromis invariance/discriminance.

2.3 Les sacs de mots

La technique des sacs de mots est l'un des outils technologiques majeur de ce projet. C'est sur lui que vont reposer l'indexation et la recherche d'images similaires.

C'est une technique issue de la recherche d'informations textuelles qui permet de décrire un ensemble de documents à partir d'un vocabulaire. Elle a été adaptée à la recherche d'images en 2003 dans l'article [SZ03]. Les documents deviennent alors des images et les mots des vecteurs descripteurs. On parle de sacs de mots visuels.

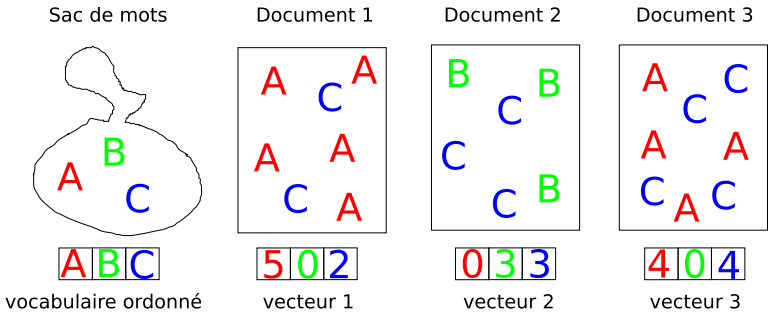


FIGURE 2.9 – Exemple de descriptions de trois documents par la technique des sacs de mots. On constate que les documents sont résumés par un vecteur de taille 3.

La figure 2.9 illustre le fonctionnement du sac de mots. On y voit en premier lieu le sac de mots, aussi appelé vocabulaire. Il contient l'ensemble des mots pouvant apparaitre dans un document. Chaque document est une combinaison de mots du vocabulaire répétés un nombre quelconque de fois. On peut donc le représenter par l'histogramme des apparitions des mots. L'histogramme prend la forme d'un vecteur de même taille que le vocabulaire, avec en i^e position le nombre d'occurences du i^e mot du vocabulaire. Cela implique que les mots du vocabulaire doivent être ordonnés, ce qui est représenté par le vecteur sous le sac de mots de la figure 2.9.

Cependant, une mesure plus précise que la fréquence d'apparition des mots à été utilisée : le tf-idf (Term Frequency-Inverse Document Frequency). Cette mesure permet d'évaluer l'importance de chaque mot dans un document, relativement à l'ensemble des documents. Elle est composée de deux coefficients : la fréquence du mot et la fréquence inverse de document.

Soit un ensemble D de N_D documents écrit à l'aide de M mots m_i, $1 \leq i \leq M$. Le i^e mot m_i apparait $n_{i,j}$ fois dans le document D_j.

La fréquence du mot Elle mesure l'importance du mot m_i dans le document D_j. La fréquence du mot m_i dans le document D_j vaut :

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

où k représente le nombre de mots distincts présents dans le documents D_j.

La fréquence inverse de document Elle mesure l'importance du mot m_i dans l'ensemble des documents de D, en donnant un poids plus important aux mots les moins fréquents. La fréquence inverse de document du mot m_i vaut :

$$idf_i = \log \frac{N_D}{k_i}$$

où k_i est le nombre de documents dans lequel m_i apparait.

La mesure tf-idf s'obtient en multipliant les deux coefficients : $tf\text{-}idf_{i,j} = tf_{i,j} \cdot idf_i$.

2.4 La recherche des plus proches voisins

Les techniques de recherche des plus proches voisins sont nombreuses. Deux vont être présentées ici : la recherche des K plus proches voisins et la recherche ε-sphère. À partir de la description d'une image requête, elles vont permettre de trouver les images les plus proches en recherchant dans l'espace de description des images. Elles vont donc être présentées appliquées à la recherche d'images, mais le principe est le même pour n'importe quel type de document.

La recherche des K plus proches voisins a pour but de retrouver les K images les plus proches de l'image requête alors que la recherche ε-sphère a pour but de retrouver toutes les images à une distance inférieure à ε. Dans le premier cas on fixe le nombre d'images à trouver et dans le second la distance maximale des images retournées.

Ces deux recherches ont besoin d'une mesure permettant d'évaluer la distance entre la signature de l'image requête et les signatures des images de la base. La mesure qui sera utilisée dans ce projet est la mesure cosine, mais d'autres mesures comme la distance de Mahalanobis peuvent la remplacer.

Pour les deux recherches, l'approche est la même : un parcours linéaire des signatures des images de la base. L'algorithme 1 page suivante des K plus proches voisins illustre les propos qui vont suivre. Celui de la recherche ε-sphère s'en déduit facilement.

Pour la recherche des K plus proches voisins, on commence donc par créer une liste L contenant les K premières signatures de la base et leur distance. L est triée par ordre croissant des distances. Notons que pour la mesure cosine, ce sont les plus grandes valeurs qui représentent les plus petites distances. On parcourt ensuite le reste des signatures de la base. Dès qu'une nouvelle signature obtient une distance inférieure à celle de la dernière signature de L, on supprime cette dernière signature et l'on insère la nouvelle de façon à ce que L soit toujours ordonnée. Lorsque toutes les signatures ont été parcourues, L contient les K signatures les plus proches de la signature de l'image

Algorithme 1 Algorithme de recherche des N plus proches voisins

```
1: MAP<STRING,SIGNATURE> index
2: SIGNATURE signature
3: LIST< Pair<STRING,FLOAT> > imagesFound
4: INTEGER nbImagesToReturn
5: Pour i = 0 → nbImagesToReturn − 1 faire
6:    imagesFound.append( Pair( index[i].key(), signature.distanceCosine(index[i].value() ) )
7: Fin pour
8: Trier la liste imagesFound
9: FLOAT currentDistance
10: INTEGER insertRank
11: Pour i = nbImagesToReturn → index.size() − 1 faire
12:    currentDistance = signature.distanceCosine( index[i].value() )
13:    Si currentDistance < imagesFound.last().second alors
14:       imagesFound.removeLast()
15:       insertRank = 0
16:       Tantque insertRank < imagesFound.size() and imagesFound[insertRank].second < cur-
          rentDist faire
17:          insertRank++
18:       Fin Tantque
19:       Si insertRank == imagesFound.size() alors
20:          imagesFound.append( Pair(index[i].key(),currentDist) )
21:       sinon
22:          imagesFound.insert( insertRank, Pair(index[i].key(),currentDist) )
23:       Finsi
24:    Finsi
25: Fin pour
```

requête.

La recherche ε-sphère est encore plus simple. On crée une liste L vide et on ajoute chaque signature dont la distance à la signature de l'image requête est inférieure à ε. L'ordonnancement de L n'est pas une obligation mais il préférable d'obtenir les images les plus proches par ordre de pertinence.

La recherche des plus proches voisins constitue l'étape finale de la recherche d'image similaires qui sera présentée à la section 2.6 page suivante.

2.5 L'indexation inversée

Une autre partie théorique importante de ce projet est l'indexation des images, présentée et analysée dans les articles [C75] et [ZMR98]. Elle permet d'accélérer la recherche des plus proches voisins, présentée à la section 2.4 page précédente, en offrant une méthode d'accès rapide aux signatures des images de la base.

Dans les explications qui vont suivre, on supposera que l'on cherche à indexer la base 600. Chaque image possède une signature. C'est un vecteur de même taille que le sac de mots. Au rang i de la signature d'une image figure le tf-idf du i^e mot du sac de mots dans cette image.

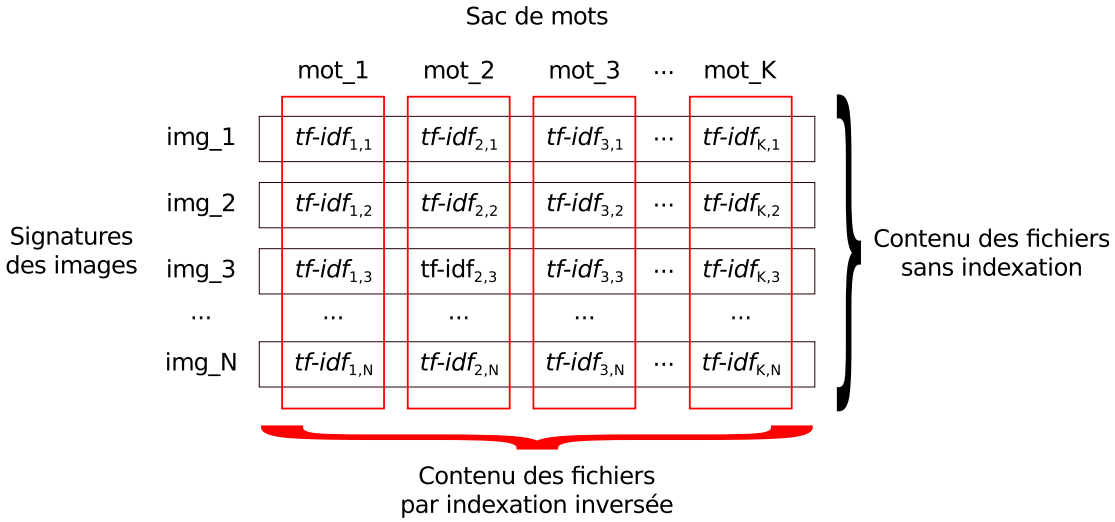


FIGURE 2.10 – Schéma d'illustration de la stratégie de l'inde inversé

Pour sauvegarder les signatures sans utiliser d'index, la solution la plus simple est de les écrire dans un fichier unique. Une amélioration rapide à mettre en œuvre est d'utiliser un fichier par signature, afin d'accéder plus rapidement à une signature voulue. C'est ce que l'on peut lire en ligne sur la figure 2.10. Chaque signature de chaque image est sauvegardée dans un fichier unique.

Le principe de l'index inversé est comme son nom l'indique d'inverser l'écriture des fichiers. Alors qu'une sauvegarde sans index associe à chaque image les tf-idf de chaque mot, l'index inversé associe à chaque mot le tf-idf de chaque image. C'est ce que l'on peut lire en colonne sur la figure 2.10.

Le premier avantage de l'index inversé apparait quand les signatures deviennent creuses, c'est-à-dire quand elles ne contiennent qu'une faible proportion des mots du sac de mots. Numériquement, cela se concrétise par un tf-idf à zéro. Plus il y aura de mots et plus les signatures seront creuses. Les fichiers créés auront une petite taille car ils ne contiendront que les tf-idf non nuls.

L'avantage de ces fichiers de petites tailles apparait lors de la recherche d'images similaires. Pour ce faire, il faut comparer la signature de l'image requête avec les signatures des images de la base. Sans index, il va falloir charger toutes les signatures. Avec l'index inversé, on ne va charger que les fichiers des mots qui apparaissent dans l'image requête. On ne reconstituera donc que les signatures des images qui ont au moins un mot en commun avec l'image requête. Il y a donc un double avantage :

- un chargement moins coûteux en temps et en nombre d'accès au disque,
- une pré-sélection des images similaires.

L'indexation inversée présente un avantage certain pour la recherche d'images similaires qui va maintenant être présentée.

2.6 La recherche d'images similaires

La recherche d'images similaire qui va être détaillée repose sur la technique des sac de mots visuels. Elle est couramment utilisée lorsque les images sont décrites par des caractéristiques locales.

Pour pouvoir réaliser une recherche d'images similaires sur une base d'images, il faut avoir préalablement calculé les signatures des images et créé le sac de mots. On peut alors interroger la base en soumettant une image requête. Les étapes de la recherche sont les suivantes :

1. Chargement du sac de mots avec élagage éventuel.
2. Extraction des points d'intérêt de l'image requête.
3. Calcul de la signature de l'image requête à partir du sac de mots (voir l'algorithme 5 page 79 de l'annexe A).
4. Suppression des mots pour lesquels le tf-idf de l'image requête vaut 0.
5. Chargement des signatures (voir l'algorithme 6 page 79 de l'annexe A).
6. Recherche des images voisines les plus proches (voir la section 2.4 page 24).

2.6.1 L'élagage du vocabulaire

L'élagage de l'étape 1 permet de ne pas prendre en compte les mots les plus fréquents. Si un mot apparait dans la plupart des images de la base, le fait de la voir apparaitre dans l'image requête n'est pas discriminant. Un nombre important de mots fréquents peut même faire baisser les performances de la recherche en rendant les images trop proches les unes des autres.

Concrètement, cette fréquence correspond à la partie *idf* du tf-idf. L'élagage consiste donc à appliquer un seuil à cette fréquence, précalculée lors de la création du sac de mots. Tous les mots ayant une fréquence d'apparition plus grande que ce seuil ne seront pas chargés.

2.6.2 La suppression des mots

Cette étape préalable au chargement des signatures ne figure pas dans les recherches d'images similaires classiques. Elle permet néanmoins d'accélérer les calculs ultérieurs en supprimant les mots dont le tf-idf avec l'image requête vaut 0.

La signature de l'image requête, calculée à l'étape 3, a une taille qui correspond au nombre de mots du sac de mots. Lors de l'étape 5, les signatures chargées auront la même taille. À l'étape 6, l'ensemble de ces signatures seront comparées à la signature de l'image requête. Le temps de comparaison va donc être proportionnel au nombre de signatures et à leur taille, qui est le nombre de mots du sac de mots.

L'étape de suppression des mots permet de réduire ce temps de comparaison des signatures en le rendant non pas proportionnel au nombre de mots du sac de mots mais au nombre de mots présents dans l'image requête. Pour cela, la signature de l'image requête est réduite à ses valeurs non nulles, ce qui revient à supprimer les mots ayant un tf-idf nul avec l'image requête. Lors de l'étape 5 de chargement des signatures, seul les tf-idf des mots encore présents dans la signature de l'image requête seront chargés. Les signatures auront donc la même taille que la signature élaguée de l'image requête. Ainsi, les comparaisons à l'étape 6 se feront sur des signatures réduites et plus elles seront petite, plus la comparaison sera rapide.

Notons que cette amélioration est valide du fait de la mesure utilisée pour les comparaisons. Il s'agit de la mesure cosine, qui se base sur le produit scalaire de vecteurs. Une valeur nulle au rang i dans l'une des deux signatures comparées implique un impact nul du rang i dans la mesure cosine. Il est donc acceptable de réduire la signature de l'image requête à ses valeurs non nulles.

2.7 Combiner plusieurs détecteurs/descripteurs

Comme nous l'avons vu aux section 2.1 page 11 et 2.2 page 18, les détecteurs/descripteurs ne décrivent pas nécessairement la même information visuelle et n'ont pas les mêmes degrés d'invariance.

L'idée de la combinaison de détecteurs/descripteurs est de mettre à profit cette complémentarité en les utilisant conjointement. Les techniques de combinaison sont nombreuses et celle qui a

été mise en place est dite « a priori ». Il s'agit de concaténer les signatures de chaque détecteur/descripteur avant d'effectuer la recherche des plus proches voisins.

Cette méthode de combinaison modifie les étapes de la recherche d'images similaires introduite à la section 2.6 page 26. Les nouvelles étapes sont les suivantes :

1. Chargement du sac de mots.
2. Extraction des points d'intérêt de l'image requête.
   Pour chaque détecteur/descripteur :
3. Calcul de la signature de l'image requête.
4. Suppression des mots pour lesquels le tf-idf de l'image requête est 0.
5. Chargement des signatures des images de la base.
   Fin pour.
6. Concaténation des signatures de l'image requête.
7. Concaténation des signatures de chaque détecteur/descripteur pour chaque image.
8. Recherche des images voisines les plus proches.

Ces étapes sont reprises sur la figure 2.11, notamment les étapes 6 et 7 de concaténation des signatures qui vont maintenant être détaillées.

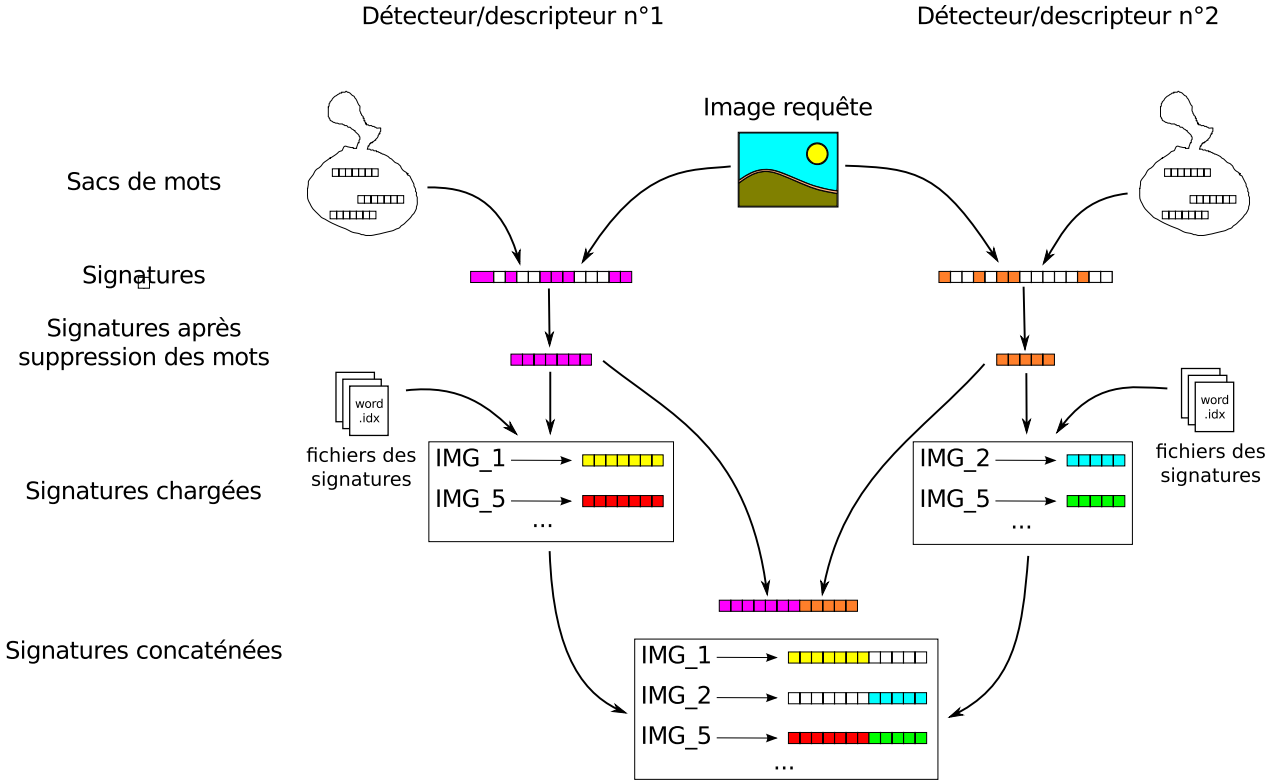


FIGURE 2.11 – Exemple de processus de recherche d'images similaires avec deux détecteurs/descripteurs illustrant la combinaison de leurs signatures.

Ces étapes consistent à concaténer les signatures de chaque détecteur/descripteur pour chaque image. Les signatures ont une taille qui dépend du détecteur/descripteur et de la suppression des mots, comme précisé à la section 2.6 page 26.

Autre fait important, si la signature d'une image I est chargée pour un détecteur/descripteur D1 et pas pour un détecteur/descripteur D2, la partie de D2 de la signature concaténée sera laissée à 0 (voir IMG_1 sur la figure 2.11). Cela signifie qu'aucun des mots de la signature de l'image requête ne figure dans la signature de l'image I obtenue avec le détecteur D2.

La concaténation pose le problème de l'homogénéisation des signatures. Suivant le détecteur/-descripteur utilisé, les valeurs peuvent être comprises dans des intervalles différents. Ainsi, D1 peut obtenir les valeurs de ses vecteurs descripteurs dans l'intervalle $[0,1]$ alors que D2 aura ces mêmes valeurs dans l'intervalle $[0,100]$. En conséquence, un détecteur avec un faible intervalle aurait un impact quasi nul sur le résultat final et l'intérêt de la combinaison serait totalement remis en question. Il est donc nécessaire de normaliser les signatures avant la concaténation. Cette normalisation donne un impact similaire à chaque détecteur/descripteur, et fournit un bon compromis des résultats de chaque détecteur/descripteur pris séparément.

Maintenant que les différents outils que sont les sac de mots, l'indexation inversée et les algorithme de recherche des plus proches voisins et de recherches d'images similaires ont été présentés, le chapitre suivant va être consacré à la base logicielle. Elle a été mise mise en place pour combiner, utiliser et évaluer les détecteurs et descripteurs détaillés au début de ce chapitre.

CHAPITRE 3

La base logicielle

3.1 Les objectifs

Les cinq premiers mois de stage ont permis de mettre au point un logiciel de recherche d'images par sacs de mots visuels, technique présentée à la section 2.3 page 23. L'objectif principal du logiciel est de pouvoir réaliser indépendamment les quatres étapes suivantes :

La détection des points d'intérêt La détection s'effectue sur une base d'images et produit la liste des vecteurs descripteurs des points d'intérêt trouvés dans les images.

La classification La classification permet de catégoriser l'ensemble des vecteurs descripteurs. Chaque classe est représentée par un vecteur et l'ensemble de ces représentants constitue le sac de mots, appellé également vocabulaire.

L'indexation Une signature est calculée pour chaque image de la base à partir du sac de mots. Ces signatures peuvent être sauvegardées en l'état, une par fichier, ou être indexées afin de rendre la recherche d'images plus rapide. Une méthode d'indexation, l'indexation inversée, est détaillée à la section 2.5 page 25.

La recherche d'images similaires Il s'agit de soumettre une image requête pour en retrouver les images les plus proches. La recherche d'images similaires, présentée à la section 2.6 page 26, nécessite donc au préalable la détection et la description des points d'intérêt de l'image requête. Deux algorithmes de recherche des plus proches voisins sont expliqués à la section 2.4 page 24.

Ces étapes constituent un ensemble de deux phases : la phase hors-ligne et la phase en-ligne.

La phase en-ligne est la recherche d'images similaires. Cette phase doit être optimisée de façon à la rendre la plus rapide possible. En effet, elle est susceptible d'être utilisée dans une application fonctionnant en temps réel. Les optimisations doivent donc être réalisées aux niveaux du nombre de lectures sur le disque et du temps de calcul.

La phase hors-ligne regroupe les trois premières étapes. Elles sont une préparation à la recherche d'images similaires. Elles doivent optimiser la taille des fichiers afin de limiter le nombre d'accès au disque lors de la phase en-ligne. Le temps d'exécution de cette phase n'est pas primordial pour la recherche d'images similaires, mais reste important pour réaliser les tests efficacement.

Après avoir présenté l'environnement de travail, le fonctionnement général des modules et leur méthode de coopération vont être expliqués. Enfin, le fonctionnement particulier de chaque module sera détaillé.

3.2 L'environnement de travail

Le stage s'est déroulé avec le matériel suivant :

- un iMac avec un processeur Intel Core 2 Duo E7600 cadencé à 3,1 Ghz, 8 Go de mémoire vive et un disque dur de 1 To,
- un NAS (Network Attached Storage) de 2 To monté en RAID 1 (Redundant Array of Independent (or inexpensive) Disks 1), ce qui représente 1 To de stockage réel.

L'iMac s'est vu doté d'un système Ubuntu 10.04, la version à support à long terme la plus récente. Le NAS a quant à lui servi à la sauvegarde des images du musée et aux sauvegardes du travail.

Les développements se sont fait en C++, avec l'apport succint du framework Qt. Ils ont été réalisés dans l'EDI (Environnement de Développement Intégré) QtCreator qui a permis une mise en route rapide et un débogage aisé.

Parallèlement au code du logiciel, des script BASH ont servi à combiner les modules et automatiser les tests. Enfin, deux logiciels libres ont été nécessaires :

- ImageMagick pour convertir les images quand le format n'était pas supporté,
- gnuplot pour réaliser les graphiques à partir des résultats des tests.

Enfin, l'optimisation des temps d'éxécution s'est faite en partie par l'utilisation de l'interface OpenMP (Open Multi-Processing). Cette interface intuitive a l'avantage de s'inclure en tant que commentaire dans un code séquentiel, évitant ainsi une duplication du code et permettant à un compilateur ne supportant pas OpenMP d'ignorer les directives. Le gain potentiel maximal est égal au nombre de cœurs logiques du processeur.

3.3 La conception des modules

Pour que les étapes décrites à la section 3.1 page précédente soient réalisables de façon indépendante, elles ont été concrétisées sous forme de modules. L'interface des modules a donc été implémentée comme une classe abstraite, avec les spécificités décrites ci-après.

3.3.1 Un double emploi

De façon abstraite, un module est un ensemble de classes pouvant être compilées comme exécutable ou comme bibliothèque. Le « ou » n'est pas exclusif : un même module peut avoir les deux fonctionnalités en même temps.

Qt a servi à concrétiser cela facilement. En effet, il propose un générateur de Makefile, qmake, qui se base sur un fichier de configuration d'extension *.pro*. Chaque module a donc été réalisé comme un nouveau projet Qt, avec un fichier de configuration pour chaque fonctionnalité.

Le fonctionnement du fichier de configuration est détaillé dans le manuel du développeur A page 74.

3.3.2 Des modules paramétrables

Ensuite, un module doit être paramétrable. Ces paramètres sont modifiables soit par les arguments de la ligne de commande lorsqu'il s'agit d'un exécutable, soit par des fonctions dédiées lorsque le module est employé comme bibliothèque. Pour faciliter et unifier leur manipulation, une classe abstraite `Module` a été conçue. Elle gère la récupération et la modification des paramètres et en offre une manipulation intuitive. Reste au développeur à définir et initialiser les paramètres.

Pour plus de détails sur les modules, se référer à l'annexe A page 74.

3.4 Les modules implémentés

Chacune des étapes de la section 3.1 page 30 a donné lieu à la création d'un module. Il y a également eu la création du module Core qui regroupe l'ensemble des classes communes aux autres modules, comme par exemple la classe `Vector`. Ce module est uniquement destiné à être une bibliothèque et ne peut en aucun cas être exécuté. Des modules de tests on également été développés et vont être détaillés après les modules principaux.

3.4.1 La communication entre modules

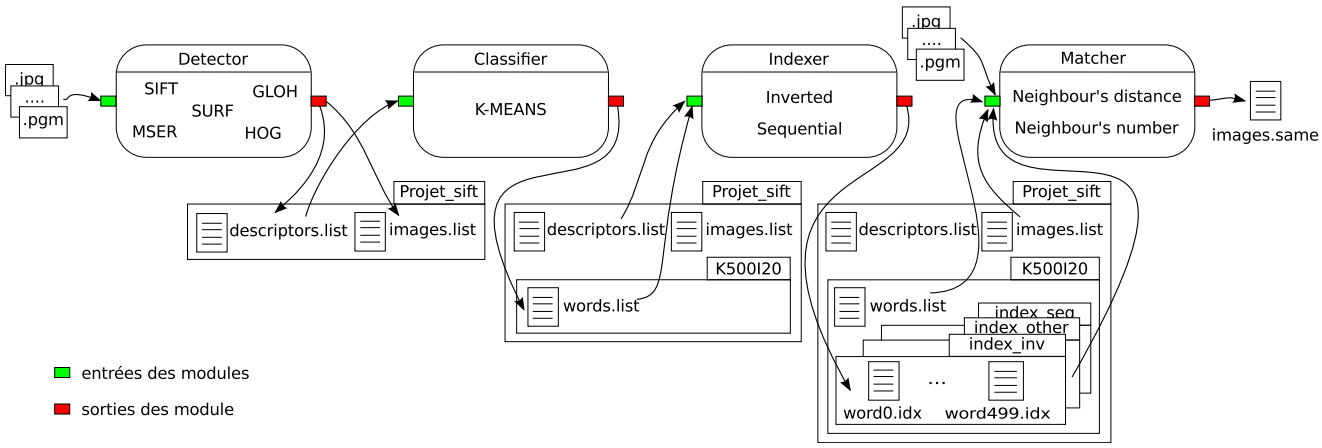


FIGURE 3.1 – Enchainement des différents modules avec les fichiers dont ils ont besoin et ceux qu'ils produisent.

Bien que s'éxécutant indépendamment, les modules sont en lien avec les résultats des modules précédents. Sur la figure 3.1, on constate que cette dépendance prend la forme de fichiers. L'objectif est de réduire le plus possible le nombre de fichiers dont dépend chaque module tout en limitant le nombre d'accès au disque dur.

Restent ainsi trois fichiers principaux :

`descriptors.list` Ce fichier créé par le module `Detector` contient l'ensemble des vecteurs descripteurs des images de la base. Il comporte un entête indiquant le nombre d'images, l'algorithme de détection/description utilisé et la taille des vecteurs descripteurs.

`images.list` Il contient la liste des noms des images de la base dans leur ordre d'apparition au sein du fichiers `descriptors.list`. Ce fichier permet de ne pas indiquer le nom des images dans le fichier `descriptors.list`. Le nom de la i^{e}image du fichier `descriptors.list` figure en i^{e}position dans le fichier `images.list`.

`words.list` Ce troisième fichier, produit par le module `Classifier`, contient les vecteurs représentant les centres des classes obtenues par classification. Chaque vecteur est un mot et l'ensemble des mots forme le sac de mots.

Il y a également un répertoire contenant les signatures des images. Celles-ci sont stockées sous la forme d'un ensemble de fichiers. Lorsqu'elles ne sont pas indexées, il y a un fichier par signa-

ture et donc par image. Mais les signatures peuvent aussi être indexées. Le nombre de fichiers dépend alors de la technique d'indexation utilisée. Pour l'indexation inversée, détaillée à la section 2.5 page 25, il y a un fichier pour chaque mot du sac de mots.

Tous les fichiers sont répartis suivant une architecture bien établie, dans un répertoire-projet, nommé Projet_sift sur la figure 3.1 page précédente. Un répertoire-projet est propre à un détecteur/descripteur pour une base d'images donnée. Il contient à sa racine les deux fichiers `descriptors.list` et `images.list`.

Le répertoire-projet peut contenir plusieurs sous-répertoires comme celui nommé K500I20 sur la figure 3.1 page précédente. Le nom K500I20 évoque une classification des vecteurs descripteurs en 500 classes sur 20 itérations. Chaque sous-répertoire contient un sac de mots, résultat de la classification : c'est le fichier `words.list`.

Tout en bas de l'architecture se situent les signatures. Elles peuvent être indexées ou non, et chaque méthode de stockage entrainera la création d'un sous-répertoire différent qui contiendra autant de fichiers que le nécessitera la méthode.

L'ensemble des modules qui vont maintenant être détaillés travaillent en coopération sur ce répertoire-projet. Son architecture tacite permet de s'affranchir du besoin de spécifier chaque fichier à chaque module : le nom du répertoire-projet suffit. Il reste cependant le choix du répertoire contenant le sac de mots lorsqu'on effectue une recherche d'images similaires. Le problème a été contourné en choisissant le répertoire le plus récemment créé ou modifié par défaut, tout en laissant la possibilité de le spécifier.

3.4.2 Le module `Detector`

Ce module a pour objectif d'extraire les points d'intérêt d'un ensemble d'images et d'en calculer les vecteurs descripteurs.

3.4.2.1 Entrées

Le module `Detector` prend en entrée le nom d'un répertoire contenant des images. Toutes les images de ce répertoire et de ses sous-répertoires sont traitées. Suivant l'algorithme de détection employé, elles seront converties à l'aide du logiciel ImageMagick dans un format compatible. L'utilisation de ce module requiert donc l'installation préalable dudit logiciel.

3.4.2.2 Sorties

Le module `Detector` produit deux fichiers :

`images.list` qui contient la liste des images traitées au format présenté sur la figure 3.2 page suivante. Si l'algorithme utilisé est à la fois détecteur et descripteur, alors TA vaut 1 et la ligne suivante contient le nom du détecteur/descripteur. S'il s'agit d'un détecteur couplé à un descripteur, alors TA vaut 2 et la ligne suivante contient le nom du détecteur puis le nom du descripteur.

`descriptors.list` qui contient la liste des vecteurs descripteurs extraits au format présenté sur la figure 3.3 page suivante.

```
N=Nombre d images
Nombre de vecteurs descripteurs
TA=Type d algorithme (1 ou 2)
TA=1 : nom du detecteur/descripteur
TA=2 : nom du detecteur
       nom du descripteur
Nom de l image 1
Nom de l image 2
...
Nom de l image N
```

FIGURE 3.2 – Composition du fichier `images.list`.

```
N=Nombre d images
Nombre de vecteurs descripteurs
Taille d un vecteur descripteur
T1=Nombre de descripteurs de l image 1
vecteur descripteur 1
...
vecteur descripteur T1
T2=Nombre de descripteurs de l image 2
...
TN=Nombre de descripteurs de l image N
vecteur descripteur 1
...
vecteur descripteur TN
```

FIGURE 3.3 – Composition du fichier `descriptors.list`.

3.4.2.3 Détails de l'implémentation

OpenMP

Ce module bénéficie des apports d'OpenMP.

En effet, l'extraction des points d'intérêt et le calcul de leurs vecteurs descripteurs sont indépendants pour chaque image. Deux images peuvent donc être traitées en parallèle. Le traitement parallèle mis en place requiert toutefois une section critique pour l'écriture des vecteurs descripteurs dans les fichiers communs `descriptors.list` et `images.list`.

Point important, le traitement parallèle implique une écriture non ordonnée des images. C'est pourquoi une seule section critique comporte l'écriture au sein des deux fichiers. Ainsi l'ordre des images reste identique dans les deux fichiers et permet une identification mutuelle.

Le processus de détection/description

On été intégrés avec succès :

- les détecteurs MSER (2.1.2.4 page 16) et SIFT (2.1.1.6 page 13)
- les descripteurs GLOH (2.2.6 page 22) et SIFT (2.2.2 page 19)
- les détecteurs/descripteurs ASIFT (2.2.4 page 21), CSIFT (2.2.5 page 21), HOG (2.2.7 page 22), SIFT (2.1.1.6 page 13 et 2.2.2 page 19), SURF (2.1.1.7 page 15 et 2.2.3 page 20) et SURF-128 (2.2.3 page 20)

Ces algorithmes implémentent une ou plusieurs des interfaces `DetectorAlgorithm`, `DescriptorAlgorithm` et `DetectorDescriptorAlgorithm`. SIFT par exemple peut être employé comme détecteur avec un descripteur tiers, comme descripteur avec un détecteur tiers ou comme détecteur/descripteur. Il implément donc les trois interfaces.

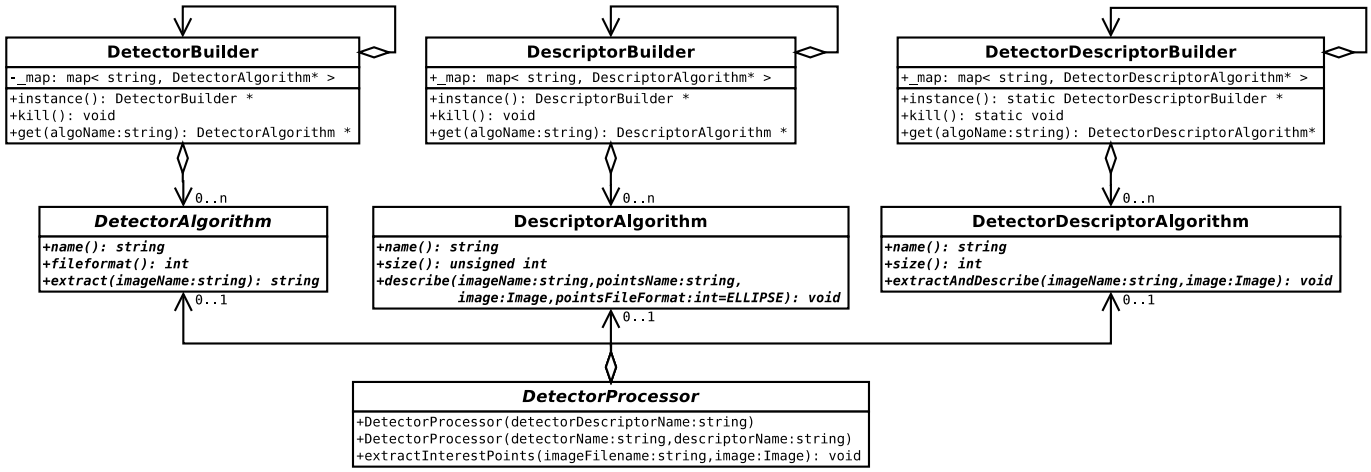


FIGURE 3.4 – Diagramme des classes intervenant dans la création d'un algorithme de détection.

Le diagramme de classes de la figure 3.4 montre comment les algorithmes sont instanciés dynamiquement. La classe nommée `DetectorProcessor` est la classe principale de déclaration et d'abstraction de ces algorithmes. Bâtie sur le modèle du patron de conception « Facade », elle est l'unique classe à utiliser pour instancier un algorithme. Comme le montre le diagramme de classes, elle construit un algorithme de détection/description soit à partir du nom d'un détecteur/descripteur, soit à partir du nom d'un détecteur et du nom d'un descripteur. Pour cela, elle appelle les classes suffixées par « Builder » qui fournissent les algorithmes. Elle s'occupe ensuite d'enchainer l'exécution des algorithmes.

Pour les détails de la création d'un nouveau détecteur, descripteur ou détecteur/descripteur, consulter l'annexe A page 74.

3.4.3 Le module `Classifier`

Le module `Classifier` a pour objectif de trouver les mots qui vont constituer le sac de mots. Il se base sur l'ensemble des vecteurs descripteurs extraits par le module `Detector` et trouve un sous-ensemble de vecteurs les représentant : ce sont les mots.

3.4.3.1 Entrées

Le module `Classifier` n'a besoin que du fichier `descriptors.list` pour fonctionner en tant qu'exécutable. Il faut lui spécifier le nom du répertoire-projet comme spécifié à la section 3.4.1 page 32.

Ce module réalise une classification et il est prévu un paramètre indiquant l'algorithme à employer. Il est également possible de passer en arguments les paramètres de l'algorithme utilisé. L'algorithme des K-moyennes a été implémenté et les paramètres possibles sont le nombre de classes et le nombre d'itérations. Pour les détails de l'implémentation, se reporter à la section 3.4.3.3 page suivante.

3.4.3.2 Sorties

Le module `Classifier` produit un fichier nommé `words.list`. Ce fichier regroupe l'ensemble des centres de classes obtenus après classification des points d'intérêt. Il se présente sous la forme présentée sur la figure 3.5 page suivante (les parenthèses n'apparaissent pas dans le fichier).

```
N=Nombre de mots
Taille d un mot
(logNNi du mot 1) (mot 1)
(logNNi du mot 2) (mot_2)
...
(logNNi du mot N) (mot_N)
```

FIGURE 3.5 – Composition du fichier `words.list`.

3.4.3.3 Implémentation de l'algorithme des K-moyennes

L'algorithme de classification des K-moyennes a été intégralement implémenté. Les objets à classer sont des vecteurs descripteurs et les centres des classes obtenues sont des vecteurs de même dimension. Les centres sont associés à la liste des vecteurs descripteurs membres de la classe.

Les choix d'implémentation ont été les suivants :

- Les vecteurs descripteurs et donc les centres de classes sont des vecteurs de type à définir à la compilation. Un type décimal est recommandé, le centre d'une classe étant calculé comme la moyenne des membres de la classe. C'est le type float qui a été retenu. Il prend deux fois moins de place qu'un double en mémoire et sur le disque, et dispose d'une précision suffisante. La possibilité de changer le type des vecteurs a cependant été conservée. Cela se fait très facilement par la déclaration d'une constante.
- Les centres des classes sont initialisés par un sous-ensemble des vecteurs descripteurs à classer. Ils sont tous différents deux à deux.
- Il n'y a pas de critère d'arrêt, uniquement un nombre d'itérations à effectuer.

Une fois les classes initialisées, l'algorithme 2 est exécuté. Ensuite, les centres des classes sont écrits dans le fichier `words.list`.

Algorithme 2 Algorithme des K-moyennes

1: **Tantque** le nombre d'itérations n'est pas atteint **faire**
2: **Pour tout** descripteur D **faire**
3: Je parcours l'ensemble des centres des classes pour trouver le plus proche de D
4: J'affecte D à la classe la plus proche
5: **Fin pour**
6: **Pour tout** classe C **faire**
7: Je calcule le centre de la classe C comme le centre des descripteurs qui en font partie.
8: **Fin pour**
9: **Fin Tantque**

Comme le suggère l'algorithme 2, le seul critère d'arrêt est le nombre d'itérations. On ne vérifie pas à la fin de chaque itération si les centres des classes ont été modifiés. C'est habituellement le critère d'arrêt principal car une fois que les centres ne sont plus modifiés, ils ne le seront plus jamais et sont considérés comme optimaux. Le nombre d'itérations n'est donc en général qu'un garde-fou pour les oscillations infinies qui empêchent les centres de se stabiliser complètement.

Il s'est avéré plus intéressant ici de ne considérer que le nombre d'itérations. Tout d'abord, vérifier que les centres ne sont plus modifiés demande un temps de calcul supplémentaire. Ensuite, les dimensions en jeu sont élevées. Ce dernier facteur augmente le nombre d'itérations nécessaires à la stabilisation complète des centres des classes. C'est pourquoi la stabilisation des centres n'est pas vérifiée.

3.4.4 Le module `Indexer`

Ce module a pour but de créer les fichiers permettant de stocker les signatures des images. Les méthodes de stockage les plus intéressantes sont les méthodes d'indexation, qui permettent d'accélérer la recherche des images similaires. Ce module propose deux méthodes de sauvegarde : sans indexation et avec indexation inversée. La technique d'indexation inversée est détaillée à la section 2.5 page 25.

3.4.4.1 Entrées

Le module `Indexer` a besoin des deux fichiers `descriptors.list` et `words.lists`, produits respectivement par les modules `Detector` et `Classifier`. Il suffit pour cela et comme pour les autres modules de lui spécifier le nom du répertoire-projet.

Il y a également un paramètre optionnel permettant de ne pas indexer les signatures. En l'absence de ce paramètre, les signatures sont indexées par la méthode d'indexation inversée.

3.4.4.2 Sorties

Le module `Indexer` produit un ensemble de fichiers d'extension *.idx*. Si les signatures ne sont pas indexées, chacune est enregistrée dans le format est présenté sur la figure 3.6.

```
(tf-idf du mot 1) (tf-idf du mot 2) ... (tf-idf du mot N)
```

FIGURE 3.6 – Format d'un fichier de sauvegarde des signatures lorsqu'elles ne sont pas indexées.

Si il y a indexation inversée, le module produit un fichier *.idx* par mot W. Le format de ces fichiers est donné par la figure 3.7.

```
N=Nombre d images qui contiennent le mot W
numero de l image 1
tf-idf de W dans l image 1
numero de l image 2
tf-idf de W dans l image 2
...
numero de l image N
tf-idf de W dans l image N
```

FIGURE 3.7 – Format d'un fichier de sauvegarde des signatures lorsqu'elles sont indexées par index inversé.

De plus, les fichiers de sauvegarde des signatures par index inversé sont écrits sous forme binaire afin de réduire le nombre d'accès au disque dur et donc le temps d'exécution. La sauvegarde sans index n'a pas bénéficié de cet avantage car elle sert uniquement de référence pour évaluer les performances de l'index inversé.

3.4.4.3 L'algorithme d'indexation

Les étapes de l'indexation sont les suivantes :

1. Chargement des points d'intérêt de l'ensemble des images de la base.

2. Chargement du sac de mots.
3. Création des signatures des images.
4. Sauvegarde des signatures avec ou sans index.

Lors de la création de leurs signatures, les images sont ajoutées comme membres des mots qu'elles contiennent. Cette manipulation est visible à la ligne 17 de l'algorithme 3. Elle simplifie l'écriture de l'index inversé qui se résume ainsi à un parcours du sac de mots.

Algorithme 3 Algorithme de création de l'index inversé

```
 1: VECTOR< WORD > bag = load_bag()
 2: VECTOR< IMAGE > allImages = load_descriptors()
 3: Pour tout IMAGE image ∈ allImages faire
 4:    FLOAT distanceMin, currentDistance
 5:    INTEGER winner, nbWordsFind
 6:    Pour tout DESCRIPTOR D_i ∈ image faire
 7:       distanceMin = bag[0].distance(D_i)
 8:       winner = 0
 9:       Pour j = 1 → bag.size() − 1 faire
10:          currentDistance = bag[j].distance(D_i)
11:          Si currentDistance < distanceMin alors
12:             distanceMin = currentDistance
13:             winner = j
14:          Finsi
15:       Fin pour
16:       image.signature[winner] += 1
17:       bag[j].addMember(image.index)
18:    Fin pour
19:    nbWordsFind = count( 0, image.signature )
20:    nbWordsFind = image.signature.size() - nbWordsFind
21:    Pour i = 0 → image.signature.size() − 1 faire
22:       image.signature[i] = (image.signature[i] / nbWordsFind) × bag[i].logNNi
23:    Fin pour
24: Fin pour
```

3.4.5 Le module `Matcher`

C'est le module le plus important. Il bénéficie de toute la préparation effectuée par les modules précédents sur la base d'images pour rechercher efficacement les images similaires à une image requête.

3.4.5.1 Entrées

Le module `Matcher` a besoin en premier lieu qu'on lui spécifie une image requête. Il a également besoin du fichier `images.list` généré par le module `Detector`, du sac de mots généré par le module `Classifier` et des signatures des images sauvegardées par le module `Indexer`. Ses fichiers sont nécessaires pour chaque détecteur/descripteur utilisé pour la recherche d'images similaires. En effet, ce module permet de combiner les résultats de plusieurs détecteur/descripteur selon la

méthode éxpliquée à la section 2.7 page 27. Les fichiers de chaque détecteur/descripteur sont retrouvés à partir des noms des répertoires-projets. L'image requête et les répertoires-projets sont les deux seuls arguments obligatoires.

D'autres arguments optionnels permettent de spécifier l'algorithme de recherche des plus proches voisins, décrits à la section 2.4 page 24, et un pourcentage d'élagage du vocabulaire.

3.4.5.2 Sorties

Il produit un fichier d'extension *.same* contenant le nom et la distance des images les plus proches de l'image requête. Les images sont ordonnées de la plus proche à la plus éloignée. Le format de ce fichier est présenté sur la figure 3.8.

```
Image requete
N=Nombre de detecteurs/descripteurs utilises
Nom du detecteur/descripteur 1
...
Nom du detecteur/descripteur N
Nom de l algorithme de recherche (number|distance) Seuil
Nombre d images similaires
Image 1 Distance avec l image requete (la plus proche)
Image 2 Distance avec l image requete
...
Image N Distance avec l image requete (la plus eloignee)
```

FIGURE 3.8 – Format du fichier d'extension `.same` produit par le module `Matcher` et contenant les images similaires à l'image requête.

3.4.5.3 Détails

Encore plus que pour le module `Detector`, l'utilisation d'OpenMP est ici prépondérante. Il s'agit du module de la partie en-ligne de la base logicielle est il doit donc être le plus optimisé pour espérer une utilisation en temps réel. L'utilisation de toute la capacité de calcul est importante et permet d'arriver plus rapidement à une utilisation temps réel sur un matériel standard.

OpenMP a donc largement été employé, notamment pour le calcul de la signature de l'image requête, étape 3 de la recherche d'images similaires présentée à la section 2.6 page 26.

3.4.6 Le module `Core`

Le module `Core` est le module de base, regroupant les outils et les objets communs à plusieurs modules. C'est actuellement le seul module non exécutable.

Les objets communs sont par exemple le vecteur, le mot, le sac de mots ou l'image. C'est dans ce module également que se trouve l'interface des modules, ce qui implique son utilisation par l'ensemble des modules. C'est aussi pour cela qu'il n'est pas un module comme les autres, ne pouvant implémenter l'interface `Module`. Il est préférable de le voir comme la bibliothèque de base des modules.

Ci-après, voici les deux outils principaux contenus dans le module `Core`.

3.4.6.1 La gestion des fichiers

La gestion des fichiers est une partie relativement importante du développement du fait du nombre important d'accès au disque dur. Pour mesurer le nombre d'accès au disque, il faut avoir

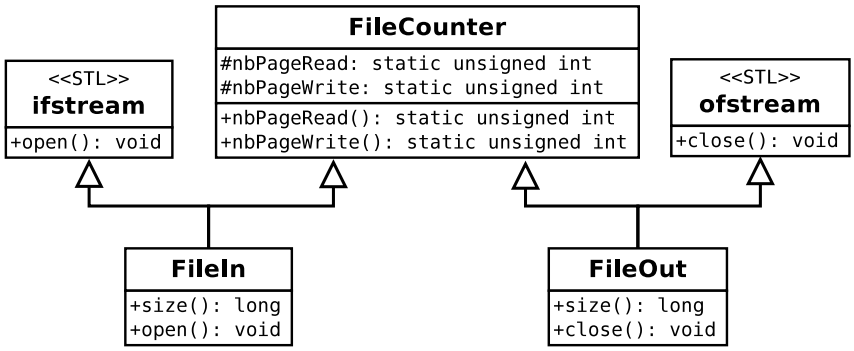


FIGURE 3.9 – Diagramme de classe des lectures et écritures de fichiers

accès aux informations concernant le nombre de lectures et d'écritures lors de n'importe quelle manipulation de fichiers. Ces informations sont quantifiées en page-disques (ou blocs), unités de lecture et d'écriture d'un disque.

Concrètement, deux compteurs statiques ont été mis en place dans la classe `FileCounter` : l'un pour le nombre de blocs lus et l'autre pour le nombre de blocs écrits.

Il sont incrémentés par les deux classes de lecture et d'écriture `FileIn` et `FileOut`. Ces deux classes respectent le patron de conception Proxy. Elles viennent se substituer aux classes de la STL (Standard Template Library), `ifstream` et `ofstream`, et incrémentent les compteurs d'accès disque lors de la manipulation de fichiers. Une classe proxy ne doit normalement hériter que d'une seule classe mais ici chacun des deux proxy hérite aussi de la classe `FileCounter`. Cela permet de cumuler toutes les informations d'accès disque dans une seule classe et ne nuit en rien à l'efficacité du patron Proxy.

Notons qu'il est donc nécessaire de passer par les deux classes `FileIn` et `FileOut` pour toute opération sur les fichiers. L'emploi direct des classes de la STL n'est pas interdit mais dans ce cas, les nombres d'accès ne seront pas comptabilisés.

3.4.6.2 La mesure des temps processeur et du nombre de blocs lus et écrits

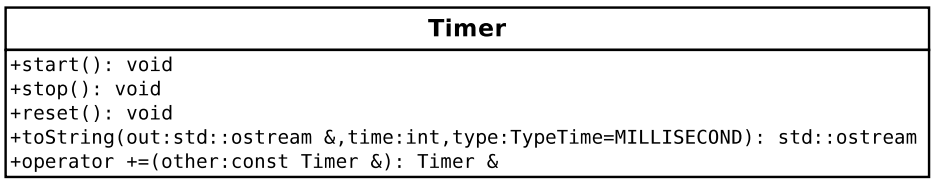


FIGURE 3.10 – Diagramme de la classe mesurant les temps processeur et le nombre d'entrées/sorties

La classe `Timer` présente sur le diagramme de classe de la figure 3.10 permet de mesurer le temps de n'importe quelle exécution. Cette classe s'utilise comme un chronomètre standard avec les fonctions publiques `start()`, `stop()` et `reset()`. Elle dispose en plus d'une fonction d'affichage qui présente les résultats obtenus sur la sortie standard.

Le timer s'utilise en s'insérant autour du code que l'on souhaite analyser. Il suffit de le créer, le déclencher, appeler les fonctions à analyser et l'arrêter. C'est ainsi qu'il a par exemple été ajouté à la fonction `exec()` de l'interface `Module`, autour de la fonction `process()` qui doit être redéfinie dans les classes filles.

Pour obtenir les informations sur les temps processeur, la classe `Timer` utilise la fonction `getrusage()` de la STL. Cette fonction permet d'obtenir les temps processeur du processus courant et/ou de ses processus fils à un moment t.

`Start()`	`Stop()`
$T_{process} = getrusage(\text{SELF})$	$T_{process} = getrusage(\text{SELF}) - T_{process}$
$T_{fils} = getrusage(\text{CHILDREN})$	$T_{fils} = getrusage(\text{CHILDREN}) - T_{fils}$
$Blocs_{lu} = FileCounter::nbPagesRead()$	$Blocs_{lu} = FileCounter::nbPagesRead() - Blocs_{lu}$
$Blocs_{écrits} = FileCounter::nbPagesWrite()$	$Blocs_{écrits} = FileCounter::nbPagesWrite() - Blocs_{écrits}$

TABLE 3.1 – Illustration des calculs du temps d'éxécution et du nombre d'accès disque par la classe `Timer`.

Le calcul des valeurs est simple et rapide. À l'appel de la fonction `start()`, des variables locales sauvegardent chacune des valeurs courantes de temps processeur et de nombre d'accès au disque. À l'appel de la fonction `stop()`, les valeurs courantes sont récupérées de la même façon, mais ce sont les différences entre les nouvelles valeurs avec les anciennes qui sont sauvegardées. Ce sont ces différences qui représentent les temps processeur écoulés et les nombres d'accès au disque.

3.4.7 Les autres modules

Un module supplémentaire permet de visualiser les images trouvées lors de la recherche d'images similaires. Le module `MatcherHtmlViewer` se base sur le fichier d'extension *.same* du module `Matcher` pour produire un fichier HTML. La figure 3.11 page suivante illustre l'affichage du fichier produit.

Enfin, un dernier module a été conçu, le module `PrecisionRecall`. Il permet de calculer les courbes de rappel/précision d'un détecteur/descripteur sur une base d'images. Le fonctionnement de ce module sera détaillé dans le chapitre 4 page 43.

La base logicielle fournit donc un ensemble de modules indépendants et cohérents. Ils permettent dans un premier temps de décrire une base d'images à partir de détecteurs/descripteurs et de préparer celle-ci à être interrogée grâce aux techniques des sacs de mots et d'indexations. C'est le module `Matcher` qui permet d'interroger efficacement la base à partir d'images requêtes en s'appuyant sur le travail préparatoire des autres modules et sur les algorithmes de recherche des plus proches voisins.

Le chapitre suivant va maintenant être consacré aux tests menés sur les détecteur/descripteur implémentés, présentés au chapitre 2 page 10. Il présentera les implémentions des détecteurs/descripteurs puis les différents résultats obtenus.

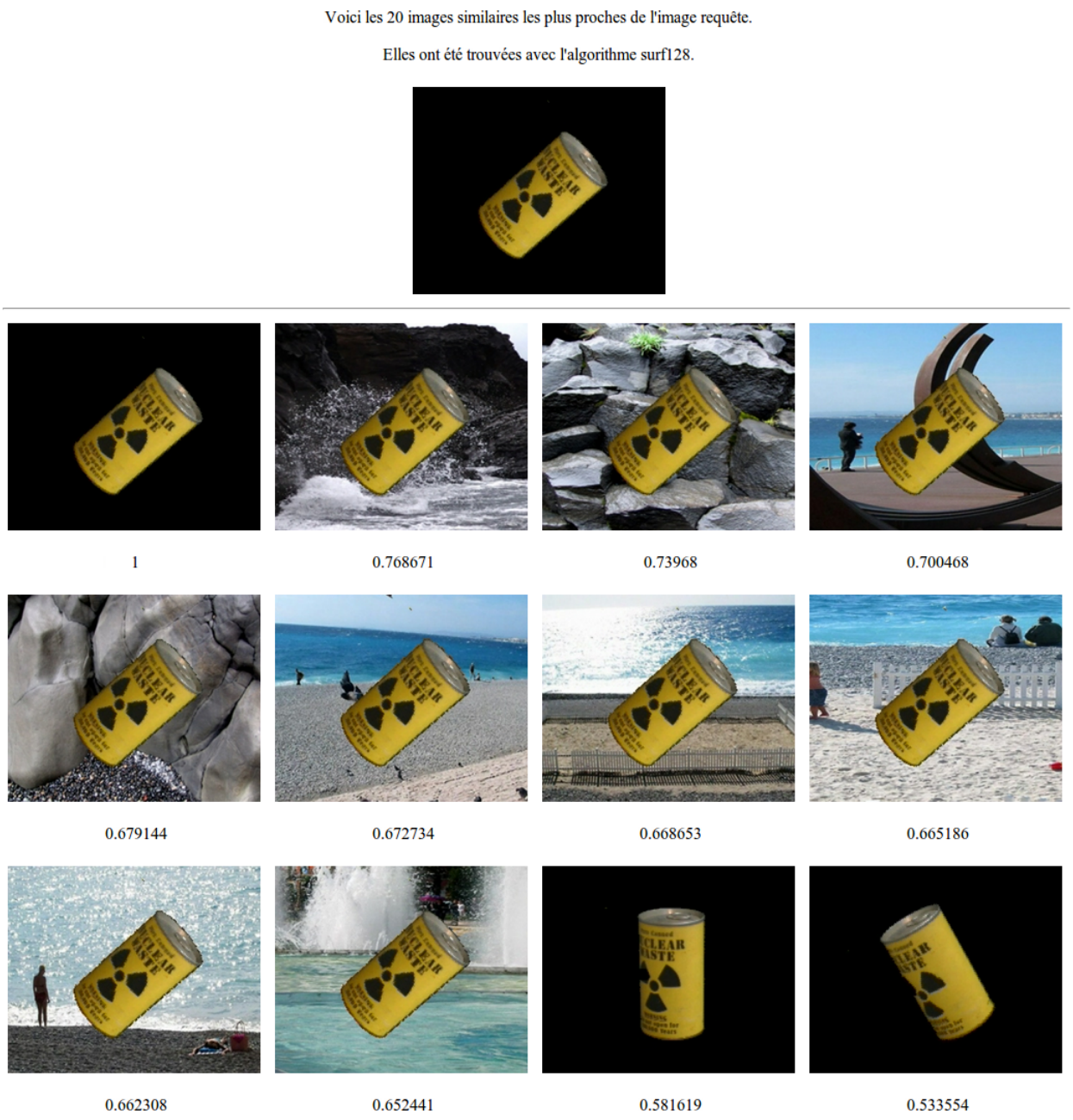


FIGURE 3.11 – Exemple de l'affichage des résultats d'une recherche d'images similaires avec le fichier produit par le module `MatcherHtmlViewer`.

CHAPITRE 4

Les expériences

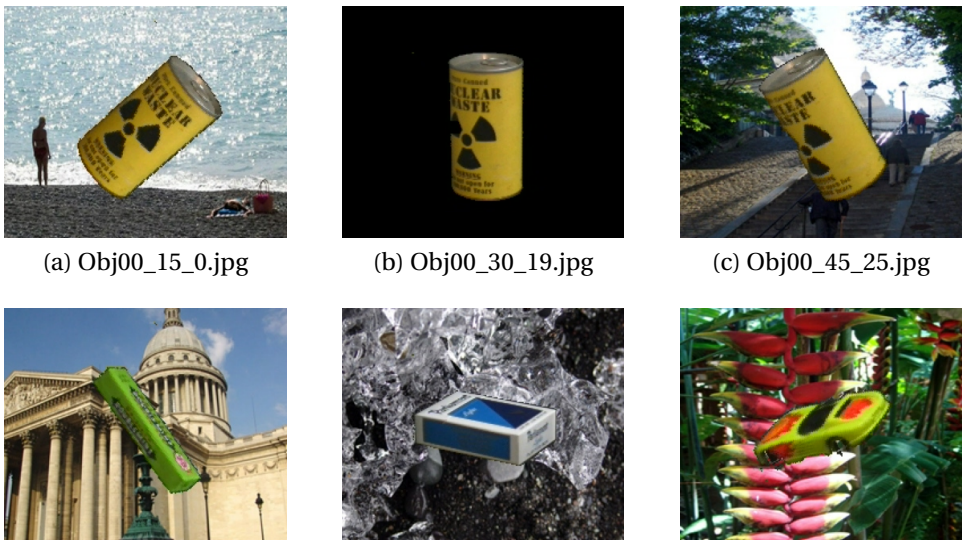
4.1 Préambule

L'ensemble des expériences menées ont été réalisées sur un sous-ensemble de la banque d'images COIL-100.

Cette base, proposée par l'Université de Columbia, comporte 100 objets pris chacun sous différents angles de vue et sur fond noir. Chaque image est disponible au format PGM ou JPG dans une résolution de 352x288 pixels.

Le sous-ensemble de test est constitué de 20 objets pris dans cette base, sous 3 angles de vue différents. Il y a 10 images pour chaque angle de vue : l'image de la base COIL-100 à fond noir et neuf autres avec des arrière-plans texturés ajoutés par photo-montage. Ces neufs images supplémentaires ont pour objectif de rendre la reconnaissance d'images plus difficile puisque l'objet est mêlé à un fond complexe différent pour toutes les images du même objet.

Il y a donc au total 600 images dont un échantillon est visible sur la figure 4.1. Nous nommerons cet ensemble d'images la « base 600 ».



(a) Obj00_15_0.jpg (b) Obj00_30_19.jpg (c) Obj00_45_25.jpg

(d) Obj02_15_67.jpg (e) Obj03_30_105.jpg (f) Obj06_45_200.jpg

FIGURE 4.1 – Exemple d'images de la base 600 créées à partir d'images de la base COIL-100

Le nommage des fichiers permet d'identifier pour chaque image l'objet représenté et l'angle de vue. Les noms sont de la forme ObjXX_YY_ZZZ, où XX est le numéro de l'objet, YY la valeur de l'angle de prise de vue (15°, 30°ou 45°) et ZZZ un numéro unique allant de 0 à 599.

4.2 Implémentations des détecteurs/descripteurs

Comme nous avons pu le voir au chapitre 3, les détecteurs/descripteurs sont au centre de la base logicielle. Le choix des détecteurs et descripteurs à intégrer s'est basé sur trois aspects :

Leur popularité Les performances de la base logicielle pourront être comparées plus facilement aux autres projets existants.

Leur complémentarité Dans l'optique de combiner les détecteurs/descripteurs, il est important de disposer de détecteurs/descripteurs complémentaire, qui ont des méthodes de détection et description différentes. Consulter la section 2.7 page 27 pour plus de détails sur la combinaison de plusieurs détecteurs/descripteurs.

Leur disponibilité Tous les détecteurs/descripteurs ne disposent pas d'implémentation disponibles et libres. Une recherche préalable des détecteurs/descripteurs disponibles a donc été effectuée avant de choisir lesquels intégrer.

Nous allons voir maintenant plus en détails les implémentations qui ont été retenues. Ce sont des exécutables obtenus sur internet, plus ou moins documentés et plus ou moins optimisés. Quand cela a été possible, c'est l'exécutable de l'auteur du détecteur/descripteur qui a été préféré.

4.2.1 SIFT

4.2.1.1 Le détecteur/descripteur

URL `http://www.cs.ubc.ca/~lowe/keypoints/`
Exécution `sift <imgIn >descriptorsFile`
Formats supportés .pgm
Description Il s'agit de la version de l'auteur de l'algorithme, David Lowe. L'exécutable n'utilise pas d'argument mais les redirections de flux. À partir d'une image, il fournit un fichier contenant la liste des points d'intérêt avec leurs coordonnées, leur échelle, leur angle et leur vecteur descripteur. Il indique en entête le nombre de points d'intérêt détectés et la dimensions des vecteurs descripteurs.

4.2.1.2 Le détecteur

URL `http://lear.inrialpes.fr/people/mikolajczyk/Database/`
Exécution `detect_points -dog -i imgIn -o pointsFile`
Formats supportés .pgm, .pnm
Description L'exécutable est proposé par Krystian Mikolajczyk, D. Lowe ayant seulement mis à disposition le détecteur/descripteur. Il contient plusieurs détecteurs dont celui de SIFT. Le fichier produit contient la liste des points d'intérêt avec leurs coordonnées, leur échelle, leur orientation et des termes précalculés comme la courbure ou le laplacien. Il comporte en entête le nombre de points d'intérêt détectés.

4.2.1.3 Le descripteur

URL `http://lear.inrialpes.fr/people/mikolajczyk/Database/`
Exécution `compute_descriptors -sift -i imgIn -p1 pointsFile -o1 descriptorsFile`
Formats supportés .png, pgm, .pnm
Description Tout comme pour le détecteur, l'exécutable est proposé par Krystian Mikolajczyk. C'est l'outil complémentaire du détecteur. Il permet également de choisir quel descripteurs utiliser. Pour calculer les vecteurs descripteurs des points d'intérêt, il se base sur le fichier produit par le détecteur. Le fichier obtenu est une copie du fichier du détecteur, au sein duquel les vecteurs descripteurs ont été concaténés à chaque point d'intérêt. Il est également possible d'obtenir ce fichier dans un second format où les valeurs précalculées sont remplacées par les valeurs a, b et c de l'équation de l'ellipse englobante :

$$a \cdot (x-u)^2 + 2b \cdot (x-u)(y-v) + c \cdot (y-v)^2 = 1$$

4.2.2 SURF et SURF-128

URL `http://www.vision.ee.ethz.ch/~surf/`
Exécution `SURF : surf -nl -i imgIn -o descriptorsFile`
`SURF-128 : surf -e -nl -i imgIn -o descriptorsFile`
Formats supportés .pgm
Description Il s'agit de l'implémentation des auteurs des algorithmes, Herbert Bay, Andreas Ess, Tinne Tuytelaars et Luc Van Gool. Cet exécutable propose de nombreux paramètres comme le nombre d'octaves ou le seuil de réponse. Il permet également d'obtenir la version étendue des vecteurs descripteurs, qui correspondent à l'algorithme SURF-128. Le fichier produit contient en entête le nombre de points détectés et la taille des vecteurs descripteurs. Viennent ensuite les points d'intérêt avec coordonnées, échelle, inclinaison, laplacien et vecteur descripteur.

4.2.3 MSER

URL `http://www.robots.ox.ac.uk/~vgg/research/affine/detectors.html`
Exécution `mser -t 2 -i imgIn -o pointsFile`
Formats supportés .png, .jpg, .jpeg, .pgm, .ppm, .tiff
Description Cette implémentation se trouve sur le site mentionné ci-dessus, maintenu par Krystian Mikolajczyk. Il n'y a cependant pas d'indication concernant l'auteur et l'origine de cette implémentation. Quoiqu'il en soit, cet exécutable est aussi très paramétrable, avec entre autres les tailles minimales et maximales d'une région et l'échelle. Les fichiers produits peuvent être de quatre formats différents. Le format le plus intérressant est celui qui comporte les valeurs de l'équation de l'ellipse. Il peut ainsi être combiné aux descripteurs utilisant l'exécutable compute_descriptors de Krystian Mikolajczyk.

4.2.4 GLOH

URL `http://lear.inrialpes.fr/people/mikolajczyk/Database/`
Exécution `compute_descriptors -gloh -i imgIn -p1 pointsFile -o1 descriptorsFile`

Formats supportés .png, pgm, .pnm
Description Comme le déscripteur SIFT, c'est l'exécutable de Krystian Mikolajczyk qui est utilisé. Les paramètres et formats de fichiers sont identiques.

4.2.5 CSIFT

URL `http://staff.science.uva.nl/~mark/downloads.html`
Exécution `colsift -colorkeys -in imgIn > descriptorsFile`
Formats supportés .ppm, pgm
Description Cette implémentation est proposée par Jan-Mark Geusebroek. Elle se base sur la version de SIFT de D. Lowe et calcule les vecteurs descripteurs comme la concaténation des vecteurs descripteurs SIFT dans chaque composante couleur. Le format de fichier est exactement le même que celui de SIFT. Notons également que J.-M. Geusebroek propose une version modifiée de l'exécutable compute_descriptors de Krystian Mikolajczyk, auquel il a ajouté son descripteur.

4.2.6 ASIFT

URL `http://www.ipol.im/pub/algo/my_affine_sift/`
Exécution `asift imgIn pointsFiles`
Formats supportés .png
Description Cette implémentation est proposée par Guoshen Yu et Jean-Michel Morel. Elle est parallélisée grâce à l'interface OpenMP. Cependant, aucun exécutable n'est fourni : le code est à disposition et un makefile permet de compiler un exécutable de demonstration. Il a fallu créer une nouvel exécutable permettant d'obtenir un fichier contenant les vecteurs descripteurs.

4.3 Les K-moyennes

4.3.1 Le protocole

L'algorithme des k-moyennes a été testé sous deux aspects : sa validité et sa stabilisation. Sa validité a été testée visuellement comme nous le verrons dans la partie 4.3.2 page suivante.

La stabilisation de l'algorithme est le point le plus délicat. L'algorithme des k-moyennes est stable à partir du moment où les centres de ses classes ne sont plus modifiés entre deux itérations successives. Le nombre d'itérations nécessaires à la stabilisation est donc capital dans cette implémentation puisque, comme cela est expliqué à la section 3.4.3.3 page 36, il est le seul critère d'arrêt.

Le test de stabilisation consiste à évaluer trois valeurs à la fin de chaque itération :

- la distance moyenne entre un centre de classe et les vecteurs membres de cette classe,
- le déplacement moyen des centres à chaque itération,
- la moyenne des différences de déplacements des centres entre l'itération précédente et l'itération courante.

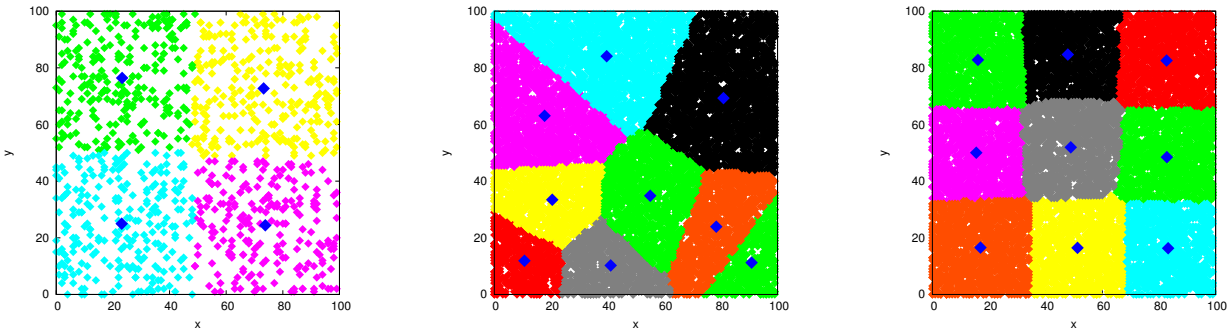
Ce test a été ajouté au code existant, dans la classe `Cluster` pour les calculs et dans la classe `KMeans` pour l'affichage sur la sortie standard. Il est cependant nécessaire de compiler le module

`Core` qui contient ces classes avec la variable *TEST_KMEANS* à vrai. Cette variable est à faux par défaut, car ces calculs ralentissent inutilement la classification en dehors des phases de tests.

Pour obtenir les graphes correspondant aux valeurs mesurées, un script Bash capture la sortie standard et en extrait les informations nécessaires à l'aide d'expressions régulières. Ce script crée ensuite un fichier par variable, qui sera affiché avec le logiciel gnuplot. Trois fichiers de configuration ont été établis pour les directives d'affichage. Notons également que chaque test est archivé dans un répertoire mentionnant la date et l'heure de l'exécution.

4.3.2 Les résultats

4.3.2.1 Les tests visuels



(a) 1000 points, 4 classes, 30 itérations (b) 10000 points, 9 classes, 3 itérations (c) 10000 points, 9 classes, 100 itérations

FIGURE 4.2 – Exemple de tests visuels efféctués sur les K-moyennes.

Les exemples de la figure 4.2 permettent de vérifier le bon fonctionnement de l'algorithme des k-moyennes.

Avec des nombres de points, de classes et d'itérations bien choisis, la figure (a) montre que les classes tendent à se répartir équitablement dans l'espace. Les figures (b) et (c) illustrent quant à elles la stabilisation de l'algorithme. En saturant l'espace de points, on finit par assimiler visuellement une classe à une surface. Comme la théorie des k-moyennes le suggère, ces surfaces constituent un diagramme de Voronoi. On constate que pour 9 classes, 3 itérations suffises rarement à obtenir un système stable, alors qu'au bout de 100 itérations, les 9 classes forment une partition homogène de l'espace.

Bien entendu, ces résultats visuels ne constituent en aucun cas une preuve de l'algorithme implémenté. Ils ont simplement permi de valider le bon comportement général de l'algorithme.

4.3.2.2 Les tests du nombre d'itérations

On vient de le constater avec les tests visuels (b) et (c) de la figure 4.2, la stabilisation de l'algorithme dépend du nombre d'itérations. Nous avons voulu tester sur les images de la base 600 le nombre d'itérations nécessaires à la stabilisation complète du système.

Comme précisé à la section 4.3.1 page précédente, trois valeurs ont été évaluées. Pour cela 605822 vecteurs à 128 dimensions ont été classés. Ils correspondent à l'ensemble des vecteurs descripteurs extraits des images de la base 600 par le détecteur/descripteur SIFT. Ce nombre de vecteurs et cette dimension sont représentatifs des vecteurs qui peuvent être obtenus par un détecteur/descripteur.

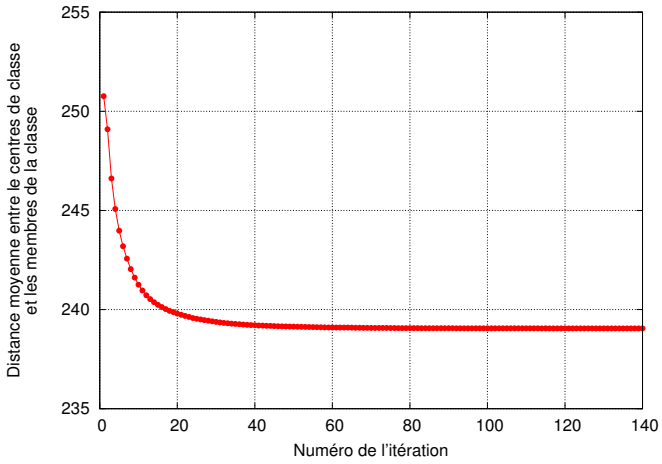
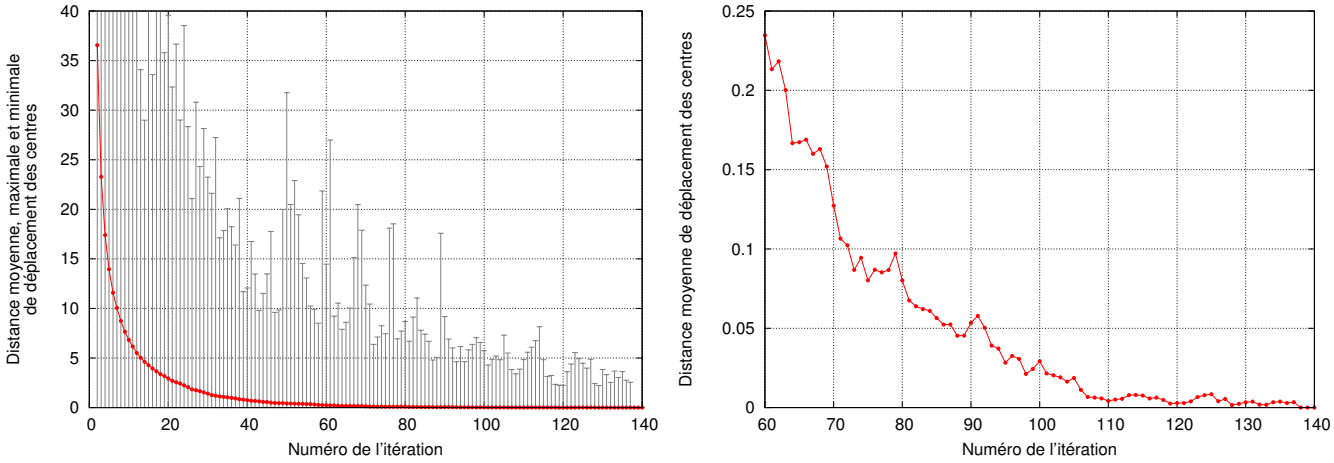


FIGURE 4.3 – Distance moyenne entre les centres des classes et leurs vecteurs membres.

La distance moyenne des centres aux membres de leur classe La distance moyenne entre chaque centre de classe et leurs vecteurs membres n'a pas apporté d'information importante. Elle a simplement appuyé l'hypothèse selon laquelle un nombre minimal d'itérations est nécessaire pour obtenir des classes dont la taille varie peu, c'est à dire quasiment stables. On peut voir sur la figure 4.3 qu'une trentaine d'itérations au moins sont nécessaires à la stabilisation.



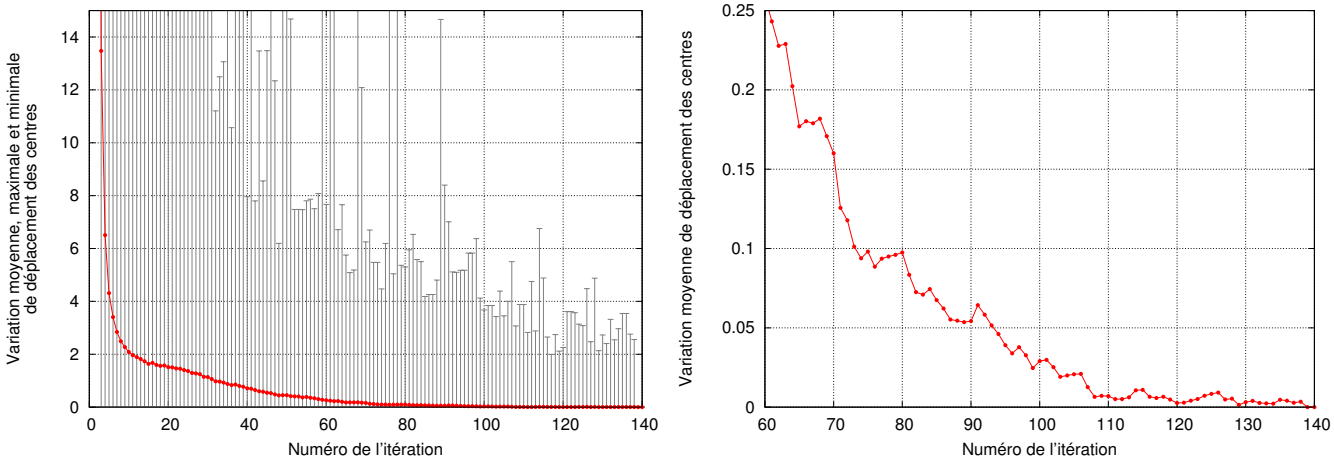
(a) Sur 140 itérations

(b) Sur les 80 dernières itérations

FIGURE 4.4 – Distance moyenne parcourue par les centres de classes entre deux itérations successives.

Le déplacement moyen des centres de classes En revanche, le déplacement moyen des centres de classes permet de déterminer le nombre d'itérations nécessaires à la stabilisation complète du système. Ainsi, on constate sur la figure 4.4 que les centres de classes ne bougent plus après 138 itérations. De plus, sur la figure (a), on constate que les centres de classes arrivent rapidement à de petits déplacements. On passe ainsi de déplacement de plus de 35 à des déplacements de moins de 1 en moins de 40 itérations, soit deux cinquièmes du nombre d'itérations nécessaires à la stabilisation complète. La figure (b) montre quant à elle la lente stabilisation au cours des quatre-vingts

dernières itérations. On peut remarquer qu'à cette échelle, la stabilisation semble un peu plus erratique mais là encore de plus en plus lente.



(a) Sur 140 itérations (b) Sur les 80 dernières itérations

FIGURE 4.5 – Différences moyenne de déplacement des centres de classes entre deux itérations successives.

La différence de déplacements des centres de classes La différence de déplacements des centres de classes de la figure 4.5 entre deux itérations successives a le même comportement que les courbes du déplacement moyen de centres de la figure 4.4 page précédente. Cela signifie que l'ensemble des centres de classes ont des déplacements de plus en plus petits. On constate cependant à la vue des valeurs maximales des la figure 4.5 (a) que certaines classes subissent encore de fortes variations à la fin des itérations nécessaires à la stabilisation.

La stabilisation complète du système a donc nécessité 138 itérations. Il ne s'agit cependant ici que d'une seule classification. Il est donc impossible de généraliser ce résultat à toutes les classifications de tous les détecteurs/descripteurs. Néanmoins, ces résultats donnent un ordre d'idée du comportement de la classification sur des vecteurs descripteurs.

4.4 Les courbes de rappel/précision

4.4.0.3 Définitions

Le rappel et la précision permettent d'estimer les performances d'un algorithme de recherche.

Considérons une base de données au sein de laquelle N_D documents sont regroupés en classes. On effectue alors une recherche de l'ensemble des documents de classe i pour obtenir N_R documents en réponse.

La rappel est alors le rapport suivant :

$$Rappel = \frac{Nombre\ de\ documents\ de\ classe\ i\ parmis\ les\ N_R\ documents\ de\ la\ réponse}{Nombre\ de\ documents\ de\ classe\ i\ parmis\ les\ N_D\ documents\ de\ la\ base}$$

et la précision le rapport suivant :

$$Précision = \frac{Nombre\ de\ documents\ de\ classe\ i\ parmis\ les\ \mathrm{N_R}\ documents\ de\ la\ réponse}{\mathrm{N_R}}$$

Leurs valeurs sont comprises dans l'intervalle $[0,1]$, 1 étant la valeur optimale. En effet, un rappel de 1 signifie que tous les documents de la classe i ont été trouvés et une précision de 1 que tous les documents trouvés sont de classe i.

Les courbes de rappel/précision qui on été utilisées pour les tests considèrent la valeur de la précision définie ci-dessus mais une valeur différente pour le rappel. Son numérateur devient juste $\mathrm{N_R}$, le nombre de documents de la réponse. Cela se justifie par la comparaison à une courbe parfaite dont le rappel aurait alors une progression linéaire et vaudrait $\mathrm{N_R}$. Il est d'usage d'utiliser cette valeur plutôt que le rappel original.

Une autre mesure permet d'évaluer un détecteur/descripteur à partir d'une unique valeur. C'est la mAP (mean Average Precision), la moyenne des précisions moyennes. Comme son nom l'indique, elle est la moyenne des précisions prisent pour chaque valeur du rappel. L'unicité de cette valeur permet de donner un classement chiffré des détecteurs/descripteurs.

4.4.1 Le protocole

Les courbes de rappel/précision ont été les tests les plus importants. Un module leur a été consacré afin de les manipuler facilement : le module `PrecisionRecall`. Le protocole de tests est entièrement corrélé au fonctionnement intrinsèque de ce module, ce qui permet de méler ci-après les présentations du module et du protocole de tests.

Ce module a besoin au minimum d'une requête et d'un répertoire-projet pour fonctionner. Les autres arguments sont donnés par l'annexe A page 74. La requête est une image ou une liste d'images. Lorsqu'il s'agit d'une image, le module fournit une courbe de rappel/précision de la requête avec cette image. Lorsqu'il s'agit d'une liste d'images, qui s'obtient à l'aide d'une expression régulière, le module produit une courbe pour chaque image de la liste, et une courbe supplémentaire représentant la moyenne des courbes obtenues pour chaque image.

D'un point de vue algorithmique, la construction des courbes de rappel/précision se déroule de la manière suivante :

1. **Initialisation du module `Matcher`** Cela évite d'allouer un nouveau module `Matcher` pour chaque image.
2. **Initialisation du graphe moyen** Les graphes ont été conçus comme des objets à part entière, pouvant s'additionner ou être divisés. Chaque graphe intermédiaire sera ajouté au graphe moyen, qui sera lui-même ensuite divisé par le nombre de graphes intermédiaires. Il contiendra ainsi leur moyenne.
3. **Calcul et écriture des graphes intermédiaires** Pour chaque image requête, une recherche des images similaires est effectuée avec l'algorithme des K plus proches voisins, détaillé à la section 2.4 page 24. Le nombre de voisins K est choisi comme le nombre d'images de même classe que l'image requête. Ce nombre doit être connu a priori. Le graphe est ensuite représenté par un tableau de taille K rempli itérativement par les précisions de $\mathrm{N_R} = 1$ à $\mathrm{N_R} = \mathrm{K}$.
   À la fin de son calcul, chaque graphe intermédiaire est écrit dans un fichier pouvant être exploité par gnuplot. Il est également ajouté au graphe moyen.
4. **Calcul et écriture du graphe moyen** Une fois que tous les graphes intermédiaires ont été créés, le graphe moyen contient leur somme. Il est simplement divisé par le nombre de graphes intermédiaires avant d'être écrit dans un fichier pour le logiciel gnuplot.

Notons que ce module ne peux obtenir que les courbes de rappel/précision d'images contenues dans la base utilisée par le répertoire-projet. La requête mentionnée plus haut doit correspondre à une ou plusieurs images du fichier `images.list` du répertoire-projet.

Bien que prévu pour la base 600, le module `PrecisionRecall` peut être employé avec d'autres bases d'images qui devront alors respecter les deux règles suivantes :

- Chaque image doit avoir un nom débutant par une chaine de caractères sans « _ » mais directement suivie de ce caractère. Cette suite de caractères doit identifier de façon unique sa classe. Tout autre image commençant par cette même chaine sera considérée comme appartenant à la classe de l'image requête.
- Une image testée doit appartenir à une classe de 30 individus. Cette condition n'est pas absolue est peut être contournée en modifiant le nombre de membres au sein du code et en recompilant le module.

Enfin, les images requêtes faisant partie de la base, chaque requête retrouve au moins une image correcte : l'image requête. Tous les graphes commencent donc avec une précision de 1.

Au final, trois choses on été testées :

- Pour chaque détecteur/descripteur, plusieurs tailles de vocabulaires ont été testées et parfois même différents nombres d'itérations pour une même taille de vocabulaire.
- Pour chaque taille de vocabulaire, deux requêtes ont été soumises :
  - Une première pour les images à l'arrière-plan noir ayant un angle de vue de 15°. Cela représente vingt images.
  - Une seconde pour l'ensemble des six-cents images.

  La comparaison de ces deux requête a permi de déterminer les détecteurs/descripteurs les plus sensibles au bruit généré par l'arrière-plan.
- La combinaison de plusieurs détecteur/descripteur complémentaires ou non

4.4.2 Les résultats

Les courbes de rappel/précision ont permi de comparer deux à deux les détecteurs/descripteurs et d'évaluer le gain de précision obtenu en utilisant différentes tailles de vocabulaires pour un même détecteur/descripteur.

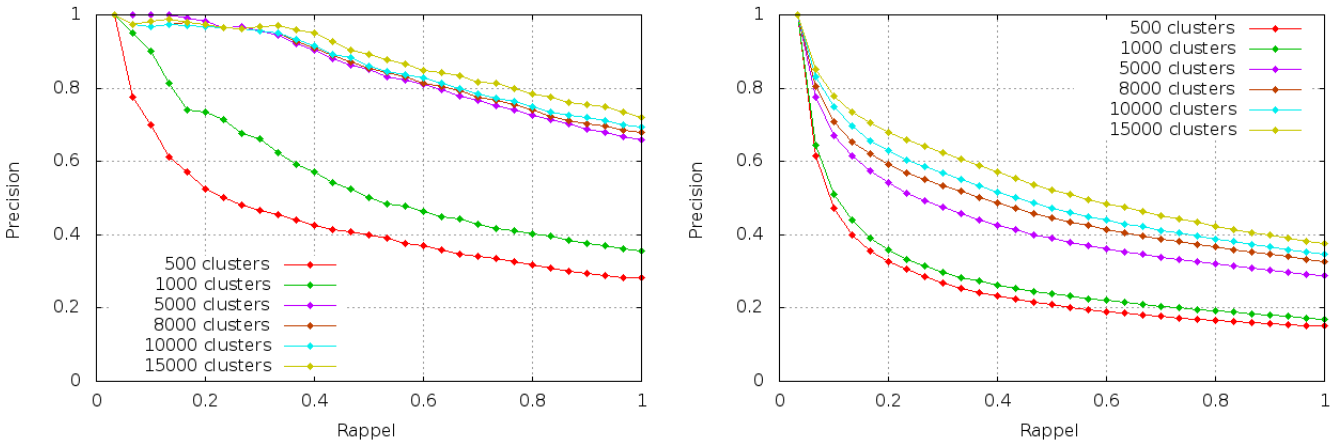
4.4.2.1 La taille du vocabulaire

Le détecteur/descripteur SIFT est le premier a avoir été intégré à la base logicielle. C'est lui qui a été utilisé pour tester des tailles de vocabulaires comprises entre 500 et 15000 mots.

On peut voir sur les graphes de la figure 4.6 page suivante les courbes obtenues sur les 20 images à fond noir et sur les 600 images.

Ce sont les images à fond noir qui obtiennent les meilleurs résultats. Cela s'explique en partie par l'arrière-plan uni où SIFT ne détecte pas de points d'intérêt. Il sont donc tous localisés sur l'objet, supprimant de ce fait le « bruit » que constituent les arrière-plans texturés.

Nous pouvons constater sur chacun des graphes de la figure 4.6 page suivante que plus le nombre de mots augmente, meilleures sont les performances. Ainsi, pour une recherche sur les 600 images, un vocabulaire de 15000 mots permet d'obtenir deux fois plus de bonnes réponses qu'un vocabulaire de 500 mots sur les trente plus proches images. Il ne faut cependant pas oublier que le temps de recherche augmente lui aussi avec la taille du vocabulaire, ce qui a son importance lorsqu'on envisage une interface utilisateur en temps réel.



(a) Sur les 20 images à fond noir

(b) Sur l'intégralité de la base 600

FIGURE 4.6 – Test de différentes tailles de vocabulaires avec le détecteur/descripteur SIFT.

Algorithme	mAP
msersift	0.645
msergloh	0.627
surf128	0.543
sift	0.470
surf	0.467
asift	0.435
siftgloh	0.430
csift	0.333
hog	0.149

(a) Sur les 20 images à fond noir

Algorithme	mAP
msersift	0.358
surf128	0.357
msergloh	0.352
siftgloh	0.315
asift	0.311
surf	0.307
csift	0.288
sift	0.274
hog	0.237

(b) Sur les 600 images

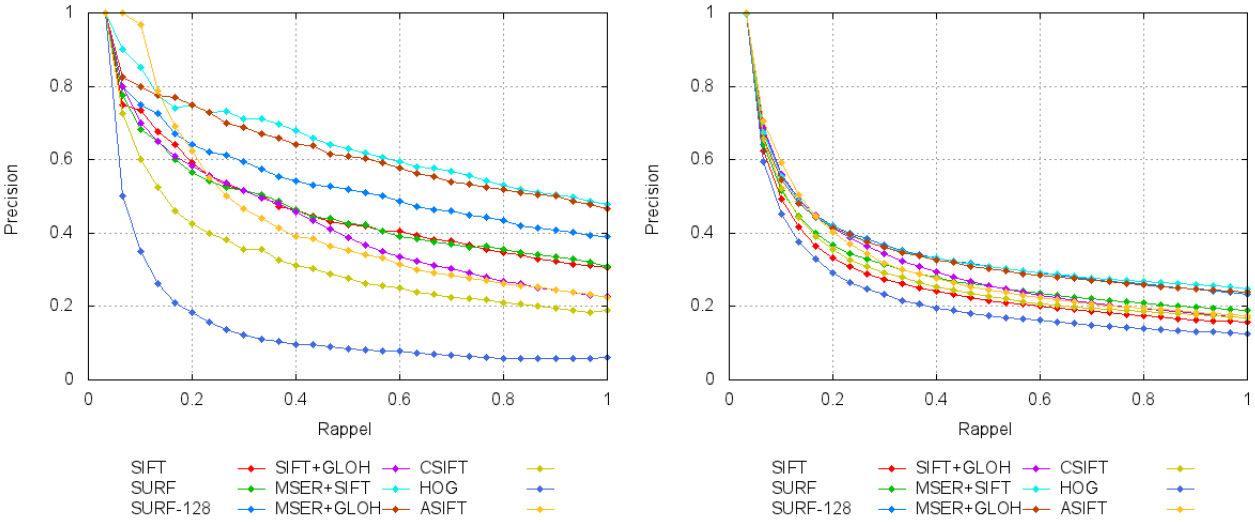
TABLE 4.1 – Précision moyenne des détecteurs/descripteurs pour un vocabulaire de 500 mots sur les 30 images les plus proches.

4.4.2.2 Comparaison des détecteurs/descripteurs

Pour comparer les performances des détecteurs/descripteurs, des vocabulaires de 500 et 5000 mots ont été utilisés. Ces deux tailles de vocabulaire ont été calculées pour chaque détecteur/descripteur avant d'obtenir les courbes de rappel/précision. Ces tests ont donc pris un temps non négligeable. Pour arriver à l'indexation de la base 600, cela peut prendre de 30 minutes pour un vocabulaire 500 mots à plus de 12 heures pour un vocabulaire de 5000 mots. Ce temps dépend également de la dimension des vecteurs produits par le détecteur/descripteur. C'est en raison de ce temps que des vocabulaires plus grands n'ont pas été testés, bien que les tests de la section précédente laissent à penser que les résultats seraient meilleurs.

Sur les courbes obtenues aux figures 4.7 page suivante et 4.8 page 54, on constate d'emblée une meilleure performance globale avec des vocabulaires de 5000 mots et avec les 20 images à fond noir. Cela confirme la tendance des précédentes courbes.

On peut noter cependant les réultats globalement décevant. Même avec 5000 mots, le meilleur détecteur/descripteur n'obtient une mAP que de 0,5. Ces résultats plutôt décevant s'expliquent par



(a) Moyenne sur les 20 images à fond noir

(b) Moyenne sur les 600 images de la base.

FIGURE 4.7 – Courbes de rappel/précision de différents détecteurs/descripteurs avec un vocabulaire de 500 mots.

Algorithme	mAP
sift	0.834
msergloh	0.802
msersift	0.795
csift	0.777
surf128	0.751
surf	0.692
siftgloh	0.640
hog	0.150

(a) Sur les 20 images à fond noir

Algorithme	mAP
siftgloh	0.572
surf128	0.562
csift	0.518
surf	0.510
msergloh	0.503
msersift	0.495
sift	0.436
hog	0.371

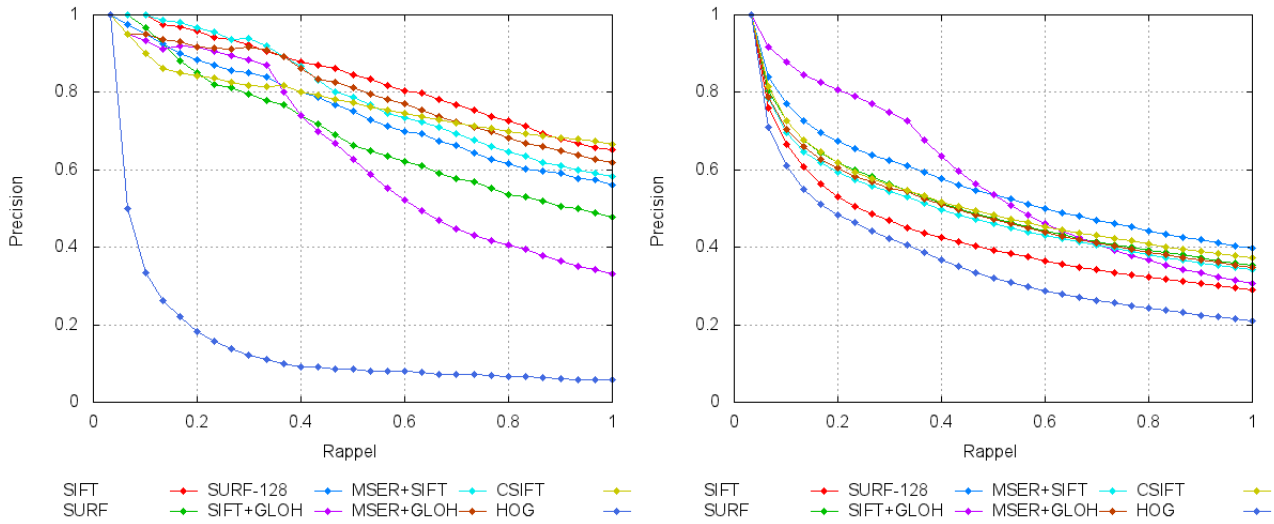
(b) Sur les 600 images

TABLE 4.2 – Précision moyenne des détecteurs/descripteurs pour un vocabulaire de 5000 mots sur les 30 images les plus proches.

les images de la base 600 qui sont des images compliquées. Les fonds texturés génerent un grand nombre de points d'intérêt avec la plupart des détecteurs/descripteurs. Les points d'intérêt des objets se retrouvent noyés au mileux des autres et la mise en correspondance devient très difficile.

Plusieurs points peuvent ensuite être soulignés :

Le moins performant Le détecteur/descripteur HOG est de loin le détecteur/descripteur le moins performant. Réputé pourtant pour être l'un des plus efficace, il échoue avec la base 600. Le problème vient du principe de HOG, qui se base sur une grille dense. Les points d'intérêt sont extraits de façon uniforme sur l'image. Or les objets de la base 600 ne constituent pas la majorité de l'image. De nombreux points d'intérêt sont extraient sur l'arrière-plan, ce qui perturbe considérablement la recherche d'images similaires. Pour les vingt images à fond noir, ce sont donc d'abord les images à fond noir qui sont trouvées les plus proches, sans distinction de l'objet représenté.



(a) Moyenne sur les 20 images à fond noir

(b) Moyenne sur les 600 images de la base.

FIGURE 4.8 – Courbes de rappel/précision de différents détecteurs/descripteurs avec un vocabulaire de 5000 mots.

SIFT et CSIFT. SIFT est moins performant que sa variante couleur CSIFT sur les 600 images avec un écart important sur le vocabulaire de 5000 mots. SIFT reste cependant plus performant sur les images à fond noir, bien que l'écart se réduise en passant de 500 à 5000 mots.

Mais il faut noter que les vecteurs descripteurs de CSIFT sont trois fois plus grands que ceux de SIFT, et cela implique deux choses. La première est que cette amélioration couleur a un coût : un temps de calcul presque de trois fois plus important. La seconde implication tient à croissance du nombre de mots qui tend à effacer l'avantage des détecteurs/descripteurs qui ont des vecteurs descripteurs de petite dimension. Cette dimension correspond la dimension de l'espace dans lequel à lieu la classification qui génère les mots. Or, pour un même nombre de vecteurs descripteurs, plus cette dimension est grande et plus les vecteurs descripteurs sont isolés. Ce fait, appelé malédiction de la dimension, entraine une proximité plus arbitraire des vecteurs descripteurs dans cet espace et donc un classification également plus arbitraire. L'impact de la malédiction se réduit avec le nombre de mots croissant, sans pour autant disparaitre. Cela explique en partie l'avantage croissant de CSIFT sur SIFT.

SURF et SURF-128 De même que pour SIFT et CSIFT, on constate que l'extension SURF-128 améliore nettement les performances de SURF. De plus, d'après leurs auteurs, les vecteurs descripteurs de SURF-128, deux fois plus grands que ceux de SIFT, sont calculés en un temps supplémentaire négligeable. Bien entendu, la recherche reste plus longue pour SURF-128, les calculs des signatures étant dépendant de la dimension des vecteurs descripteurs.

SIFT+GLOH. Le comportement le plus surprenant revient à la combinaison de détecteurs/descripteurs SIFT+GLOH. La mAP de la figure 4.2 page précédente indique que SIFT+GLOH est le détecteur/descripteurle plus performant avec une précision moyenne de de 0,572 pour sur 30 images retournées.

On constate effectivement sur la figure 4.8 que ses performances sont remarquables sur les 10 images les plus proches où il obtient plus de 10% de précision supplémentaire sur le second détecteur/descripteur le plus performant. Par contre, ses performances s'écroulent brutale-

Algorithme	mAP
surf128+msersift	0.843
msersift+msergloh	0.843
sift	0.834
surf128+msergloh	0.829
siftgloh+msergloh	0.792
siftgloh+msersift	0.782
surf128+siftgloh	0.699

(a) Sur les 20 images à fond noir

Algorithme	mAP
surf128+msergloh	0.572
siftgloh	0.572
surf128+msersift	0.568
surf128	0.562
siftgloh+msergloh	0.556
surf128+siftgloh	0.552
siftgloh+msersift	0.550
msersift+msergloh	0.529

(b) Sur les 600 images

TABLE 4.3 – Précision moyenne des combinaisons de deux détecteurs/descripteurs pour deux vocabulaires de 2500 mots sur les 30 images les plus proches.

ment au-delà.

Une explication possible est la prépondérance de la détection de l'orientation. En effet, plusieurs essais ont montré que les premières images les plus proches sont généralement les images contenant le même objet avec la même orientation. De plus, les images identifiées comme proches alors qu'elles contiennent un autre objet le sont visuellement au niveau de l'arrière-plan. Ce couple de détecteurs/descripteurs ne serait pas invariant aux rotations mais l'un des plus invariants aux occultations. On voit sur la figure 4.8 page précédente la courbe chuter brutalement après un rappel de 10/30, c'est-à-dire une fois que les dix images avec le même objet dans la même orientation ont été retrouvées.

4.4.2.3 Combinaisons de détecteurs/descripteurs

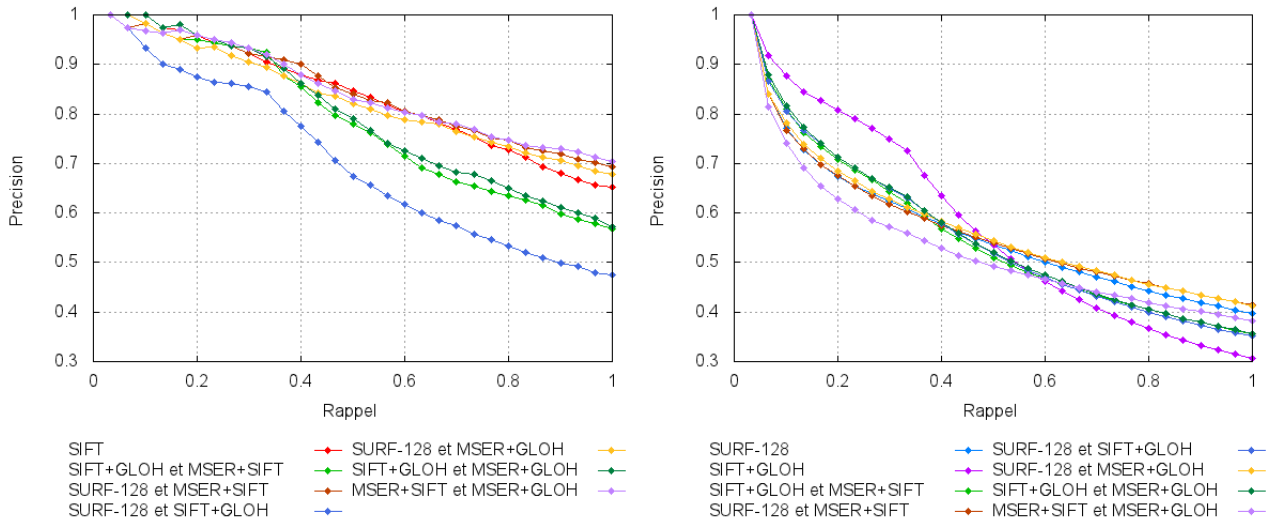
Présentée à la section 2.7 page 27, la combinaison de détecteurs/descripteurs doit permettre d'associer plusieurs détecteurs/descripteurs lors de la recherche d'images similaires. En choisissant des détecteurs/descripteurs complémentaires, nous espérons obtenir de meilleurs résultats qu'avec chaque détecteur/descripteur pris séparemment.

Au niveau de l'implémentation, la base logicielle peut combiner un nombre quelconque de détecteurs/descripteurs. L'exécutable ne prévoit pour l'instant deux détecteurs/descripteurs au plus en arguments car aucun test n'a été effectué plus de deux détecteurs/descripteurs.

Pour pouvoir comparer les performances des combinaisons aux résultats obtenus individuellements par chaque détecteur/descripteur, les tests ont porté sur des vocabulaires de 2500 mots. Un vocabulaire combiné aura ainsi une taille de 5000 mots et les résultats pourront être comparés aux mAP du tableau 4.2 page 53 et aux courbes de la figure 4.8 page précédente. On peut en effet estimer que les temps de recherche sur un vocabulaire de 5000 mots ou deux vocabualire de 2500 mots sont du même ordre, en raison de l'algorithme de recherche utilisé et détaillé à la section 2.6 page 26.

Six combinaisons ont pour l'instant été testés, avec des objectifs différents. Les résultats de ces tests figurent sur le tableaux des mAP 4.3 et sur les graphes de la figure 4.9 page suivante. Attention, la précision des deux graphes ne commence qu'à 0,3 et l'allure des courbes ne peut être comparée directements à l'allure des courbes de la figure 4.8 page précédente. Le meilleur détecteur/descripteur a donc été ajouté : SIFT sur les 20 images à fond noir et SIFT+GLOH sur les 600 images.

Le comportement de SIFT+GLOH étant atypique, SURF-128 a également été ajouté pour comparer l'allure générale des courbes.



(a) Moyenne sur les 20 images à fond noir

(b) Moyenne sur les 600 images de la base.

FIGURE 4.9 – Courbes de rappel/précision de combinaisons de deux détecteurs/descripteurs avec des vocabulaire de 2500 mots.

SURF128/MSER+SIFT La combinaison SURF-128/MSER+SIFT est l'une des deux combinaisons a obtenir une mAP plus élevée que SIFT sur les 20 images à fond noir. On constate cependant sur le graphe (a) de la figure 4.9 que la combinaison SURF-128/MSER+GLOH obtient également de meilleurs résulats que SIFT à partir d'une vingtaine d'images retournées. Ces deux résulats laissent donc à penser que les points obtenus par des détecteurs de type SIFT comme SURF-128 et les points obtenus par MSER sont bien complémentaires.

SIFT+GLOH On constate également sur les 20 images que les combinaisons avec SIFT+GLOH sont les plus mauvaises. Pour deux d'entre elles, SIFT+GLOH/MSER+GLOH et SIFT+GLOH/MSER+SIFT, les premières précisions sont pourtant les meilleures. Il semblerait donc que les faibles performances de SIFT+GLOH à partir de 10 images retournées ne suffisent pas à compenser l'avantage donné sur les 10 premières dans une combinaison.

MSER+SIFT/MSER+GLOH Cette combinaison permet de regarder ce que donne la combinaison de deux descriptions différentes de mêmes points d'intérêt. On constate sur le tableau de mAP que cette combinaison est la plus performantes en moyenne sur les 30 première images retournées lorqu'on soumet les 20 images à fond noir. Ses performances sont à l'inverse les plus faibles sur 600 images. Ce constat surprenant est à modérer car si pour 600 images les résultats sont les plus mauvais sur les 20 premières images retournées, MSER+SIFT/MSER+GLOH arrivent en milieux de classement lorsque 30 images sont retournées.

Les combinaisons de détecteurs/descripteurs ne permettent donc pas nécessairement d'obtenir de meilleurs résultats. Cependant, on constate que sur les 600 images, même la combinaison MSER+SIFT/MSER+GLOH qui a la mAP la plus faible, se classerait troisième parmis les détec-

teurs/descripteurs individuels. La combinaison semble donc un outil puissant pour l'amélioration des performances.

4.5 Les mesures de temps processeur et du nombre d'accès au disque dur

4.5.1 Le protocole

Les mesures des temps d'utilisations du processeur et du nombre d'accès au disque dur ont été effectuées pour la phase de recherche d'images similaires décrite à la section 2.6 page 26. C'est la phase en-ligne et elle doit être la plus proche possible d'une utilisation en temps réel. Les mesures ont permi d'évaluer l'impact d'éventuelles modifications apportées.

Cela a donné lieu à la création de la classe `MatchTestTimerClust` qui produit un exécutable. Il faut lui spécifier le nom du répertoire-projet, le nombre de mots du sac de mots et le nombre de recherches à réaliser. Il effectue autant de recherches d'images similaires que spécifiées et obtient les mesures pour chacune à l'aide d'un timer. Ces mesures sont affichées sur la sortie standard dans un format compatible avec gnuplot. Le calcul des valeurs moyennes est réalisé en additionnant les valeurs du timer de chaque recherche à un timer tiers puis en divisant ce timer par le nombre de recherches.

Attention, seules les étapes 4 à 6 de l'algorithme de la section 2.6 page 26 ont été prises en compte. En effet, dans une interface utilisateur qui permettrait d'interroger la base d'images, les étapes précédentes pourraient être réalisées une seule fois. Seules les étapes 4 à 6 seraient répétées pour une recherche effective. Le module `Matcher` a été modifié en conséquence car lui seul peut accéder aux détails des étapes de l'algorithme. Ces modifications n'ont pas altéré le fonctionnement général du module. Un timer a simplement été ajouté en paramètre optionnel des fonctions concernées. Les timers sont donc créés par la classe de tests `MatchTestTimeClust` et passés en argument des fonction de la classe `Matcher` qui les démarre puis les arrête entre les étapes 4 et 6.

L'objectif de ces mesures est de tester l'impact du nombre de mots et également de comparer les performances de l'index inversé avec l'index séquentiel. Un script Bash permet d'automatiser ces tests. Pour un type de recherche donné, séquentielle ou avec index inversé, il exécute `MainTestTimeClust` avec plusieurs nombres de mots différents. Il combine ensuite les sorties de ces exécutions pour produire un fichier qui sera affiché par gnuplot.

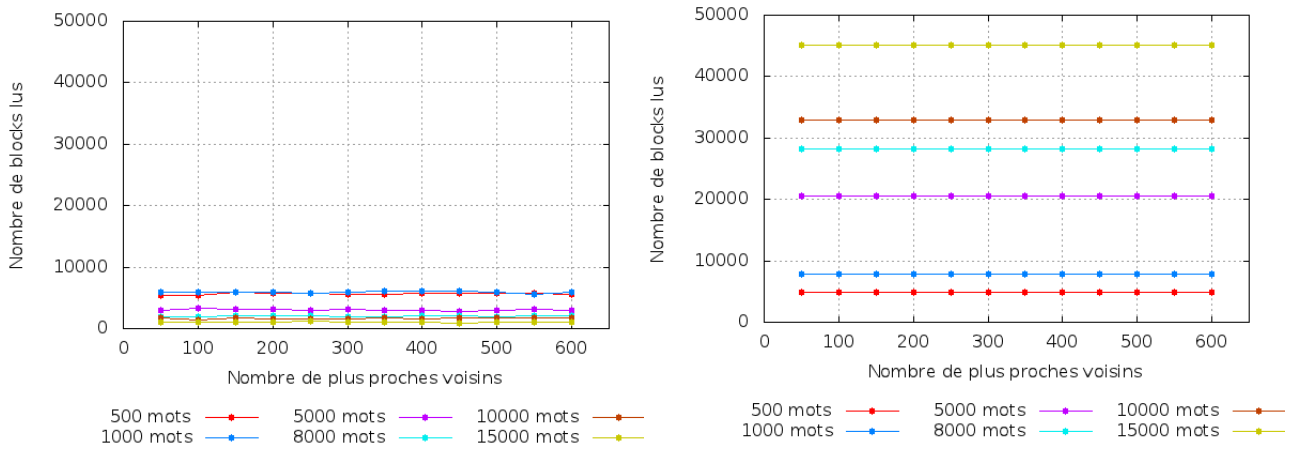
4.5.2 Les résultats

4.5.2.1 Impact du nombre de plus proches voisins

Il s'agit de mesurer l'impact que peut avoir le choix du nombre de plus proches voisins à retourner sur le nombre d'accès au disque dur et sur le temps d'occupation du processeur. Ces mesures ont été effectués avec les méthode de recherche implémentées : la recherche séquentielle, sans index, et la recherche avec index inversé. Elles ont de ce fait permis une première comparaison des performances de chacune des méthodes.

Au final, que ce soit pour le nombre d'accès au disque dur ou pour le temps d'utilisation du processeur, le constat est le même :

- le nombre de plus proches voisins n'influe pas sur les performances
- la recherche est bien plus performante avec un index inversé que sans index.



(a) Avec un index inversé

(b) Sans index

FIGURE 4.10 – Nombre d'accès disque lors d'une recherche d'images similaires en fonction du nombre de plus proches voisins.

Le nombre d'accès au dique dur. Les légères fluctuations que l'on peut observer sur le graphe 4.10 avec un index inversé sont dûes au nombre de mots présents dans les 100 images tirées aléatoirement parmis les images de la base 600. Le nombre moyen d'accès au disque pour un nombre de plus proches voisins recherchés k vaut :

$$\mathrm{D}_k = \frac{\sum_{i=1}^{100} \sum_{m=1}^{n_i} sizeof(wordFile_m)}{100}$$

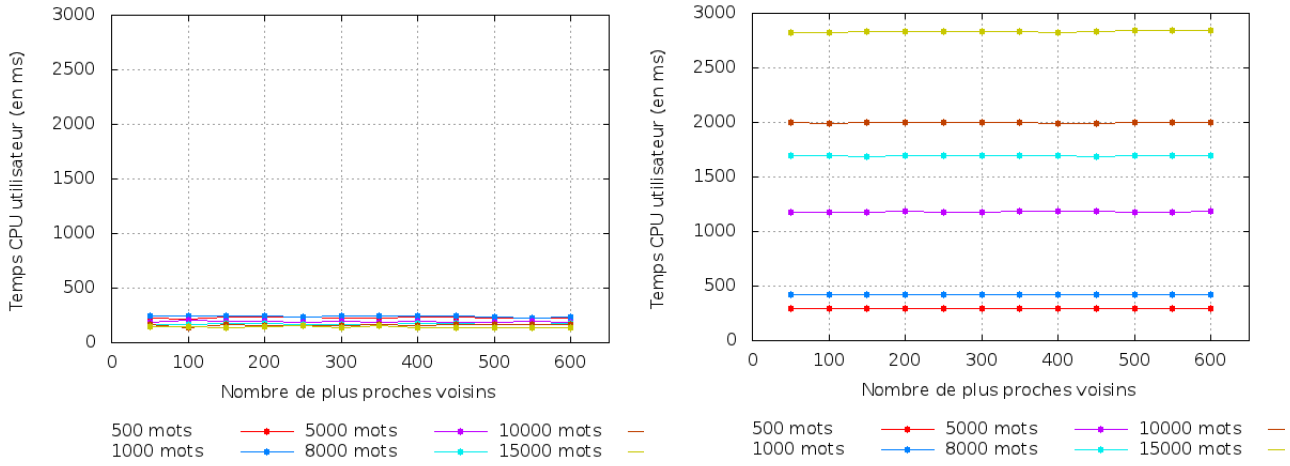
où n_i est le nombre de mots représentés dans l'image i.

Cette formule permet de constater que le nombre d'accès au disque est bien indépendant de k.

Les fluctuations évoquées ci-dessus ne sont pas observées lors d'une recherche séquentielle car tous les mots sont lus quelque soit la requête. Le nombre d'accès au disque dur est donc constant pour un nombre de mots donnés.

Enfin, ces statistiques permettent de constater l'avantage incontestable de l'index inversé sur la recherche séquentielle : si pour 500 mots le nombre d'accès au disque dur est identique pour les deux méthodes, il est 45 fois plus élevé pour une recherche séquentielle sur un vocabulaire de 15000 mots !

Le temps d'utilisation du processeur. Concernant le temps d'utilisation du processeur, le même raisonnement peut être mené. La similitude des graphes de la figure 4.11 page suivante avec les graphes de la figure 4.10 est frappante : à l'échelle près, les courbes ont un comportement identique. On constate cependant, à la différence du nombre d'accès au disque dur, de légères fluctuations de l'ordre du pourcent lors des recherches séquentielles. La raison repose là encore sur nombre de mots présents dans les images tirées aléatoirement. Plus la somme des mots présents dans toutes les images tirées aléatoirement est grande, plus la recherche des plus proches voisins est longue. En effet, le nombre de mots présents détermine la taille des vecteurs descripteurs qui seront comparés. Cela en raison de l'étape 4 de l'algorithme de recherche d'images similaires présenté à la section 2.6 page 26 qui réduit les vecteurs descripteurs aux mots qui apparaissent dans l'image requête.

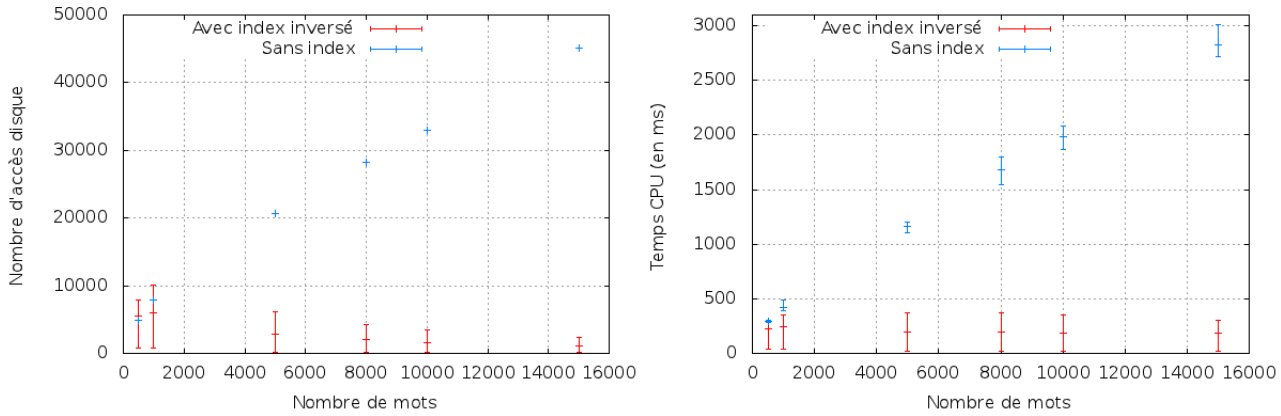


(a) Avec un index inversé (b) Avec un index séquentiel

FIGURE 4.11 – Temps d'utilisation du processeur par une recherche d'images similaires en fonction du nombre de plus proches voisins.

4.5.2.2 Impact du nombre de mots

À l'inverse du nombre de plus proches voisins, le nombre de mots influence les performances de la recherche d'images similaires.



(a) Nombre d'accès au disque en nombre do blocs (b) Temps d'utilisation du processeur en ms

FIGURE 4.12 – Nombre d'accès au disque et temps d'utilisation du processeur moyennés sur 100 recherches en fonction du nombre de mots du sac de mots.

Le nombre d'accès au dique dur. Le nombre d'accès au disque dur augmente proportionnellement à la taille des fichiers où sont enregistrées les signatures.

Pour une recherche séquentielle, la taille moyenne des fichiers augmente avec le nombre de mots puisque chaque fichier contient la liste des mesures tf-idf d'une image pour chaque mot du sac de mots. On constate sur la figure 4.12 (a) une croissance linéaire des accès au disque dur lors d'une recherche séquentielle.

À l'inverse, avec un index inversé, le nombre d'accès au disque dur diminue lentement quand le nombre de mots augmente. Ce comportement résulte d'un fait important : la taille des fichiers d'index diminue quand le nombre de mots augmente.

En effet, bien que le nombre de mots augmente, le nombre de vecteurs descripteurs extraits par un détecteur/descripteur pour une image donnée ne varie pas. Cela implique que plus il y a de mots, moins il y a de vecteurs descripteurs associés à un mot et donc moins il y a d'images associèes. Par contre, une même image se trouve associée à un nombre croissant de mots. De ce fait, les accès au disque dur sont soumis à deux contraintes :

- le nombre de mots associés à l'image requête qui augmente,
- la taille des fichiers représentant ces mots qui diminue.

C'est le second facteur qui prend le dessus sur le premier, et même largement. C'est pourquoi le nombre d'accès au disque dur diminue malgré l'augmentation du nombre de fichiers.

Le temps d'utilisation du processeur. Lors d'une recherche séquentielle, de même que pour le nombre d'accès au disque dur, le temps processeur augmente lorsque le nombre de mots du sac de mots augmente. C'est le nombre de mots qui est directement responsable de cette augmentation. En effet, lors d'une recherche séquentielle, les signatures des images ont une taille égale au nombre de mots. Le temps d'utilisation du processeur est dû en grande partie à la comparaison des signatures des images avec la signature de l'image requête et l'on constate sur la figure 4.12 page précédente (b) que le temps global augmente quasi-linéairement.

Avec un index inversé, le temps processeur n'augmente pas et diminue même très légèrement. En effet, contrairement à une recherche séquentielle, plus il y a de mots, moins il y a d'images chargées à partir de l'index. Bien que les temps de calculs de chaque image soient de plus en plus long, le temps de calcul global n'augmente pas.

L'impact du nombre de mots sur les performances de la recherche d'images similaires valide donc largement l'abandon de la recherche séquentielle et l'utilisation d'un index inversé.

4.6 Les perspectives

4.6.1 Bilan

Si les expériences menées sur l'algorithme des k-moyennes ont permi d'établir d'affiner les paramètres de construction de la base, les résultats importants pour la suite sont les expériences menées sur les détecteurs/descripteurs.

Ces expériences ont montré ainsi que la taille du sac de mots n'impacte pas sur le temps de recherche d'images similaires en ligne lorsqu'est utilisé un index inversé. De ce fait, le nombre optimal de classes pour la classification des vecteurs descripteurs peut être utilisé sans crainte de pertes de performances. Les courbes de rappel/précision ont hiérarchisé les détecteurs/descripteurs par leurs performances sur la base 600. S'il reste à en tester certains comme le détecteur de symétries, ils pourront immédiatement se positionner dans cette hiérarchie.

4.6.2 Ce qu'il reste à tester

4.6.2.1 La classification

Les expériences par exemple ont été menées sur des classifications de vingt itérations. La section 4.3.2 page 47 a montré qu'environ cent-cinquante itérations sont nécessaires à la stabilisation complète. On peut donc estimer que les performances seraient supérieures et donneraient des résultats encore plus pertinents avec une classification stabilisée.

On pourrait également tester un autre méthode de classification. En effet, si toutes sont censés donner une classification optimale, les temps de classification ne sont pas les mêmes. Or nous avons pu constater que les temps de classification par les K-moyennes sont déjà conséquents sur 600 images. Bien qu'il s'agisse d'une phase hors-ligne, une classification plus rapide serait la bienvenue pour les trois-cents mille images du musée.

4.6.2.2 Les détecteurs/descripteurs

Il reste à implémenter d'autres détecteurs/descripteurs :

- le descripteur MSCR qui devrait améliorer les résultats actuels de MSER,
- le détecteur de symetries qui devrait apporter de nouveaux points d'intérêt assez différents des autres détecteurs,
- des détecteurs et descripteurs qui n'ont pas fait ici l'objet d'études comme SUSAN, FAST, DAISY, etc.

De même, les combinaisons de plusieurs détecteurs/descripteurs n'ont été toutes testées. Il serait intéressant de continuer dans cette voie car les résultats semblent plutôt interessants. Cette méthode pourrait permettre de mieux cibler les invariances lors d'une détection.

4.6.3 Les photographies du musée

Il n'en reste pas moins que l'efficacité des détecteurs/descripteurs reste à prouver sur les photographies du musée.

Pour ce faire, Sylvain Besson, responsable du service inventaire-documentation du musée, a fourni un ensemble de photographies avec différents scénarios. Le scénario les plus complet concerne les photographies du photographe André Kertész parues dans les revues « Art et Medecine » et « Vu » de 1931 à 1936. Les photographies originales et celles publiées ont été mises en correspondance « à la main » et constituent donc une bonne base de tests. Deux autres scénarios avec les photographies de Erwin Blumenfeld et André Steiner sont disponibles pour tester les détecteurs/descripteurs. On peut voir un échantillon de ces scénarios sur la figure 1.1 page 8.

4.6.4 Un modèle de description paramétrable

L'objectif final est d'obtenir une application paramétrable capable de mettre en correspondance des images selon différents critères. Il sera donc important de classer les couples de détecteurs/descripteurs suivant leurs invariances et de trouver les combinaisons capables de répondre au mieux aux critères de recherche.

Il serait par exemple intéressant de créer un système de notation prenant en compte les invariances, la discriminance et l'efficacité de chaque détecteur/descripteur. Ce système serait alors capable en association des détecteur/descripteurcapables de répondre à des requêtes à partir de critères compréhensibles par n'importe quel utilisateur : les couleurs, les formes, le point de vue, etc.

L'application serait alors à même de répondre aux demandes de l'utilisateur de manière ciblée et précise.

Conclusion

Ce stage a permis de mettre au point un ensemble de modules capables de réaliser une recherche d'images similaires dans un corpus d'images.

Cette recherche se base sur des techniques de description locale des images. Elles débutent par la détection de points d'intérêt dans les images qui sont ensuite décrits. L'objectif de ces techniques est de fournir une description invariante aux changements pouvant survenir entre deux images représentant le même objet ou la même scène. La description doit cependant rester suffisamment discriminante pour que la similarité soit suffisamment significative.

Pour être le plus efficace possible lors de la recherche, certains modules sont chargés de préparer la base à l'interrogation. Ils utilisent pour cela des outils puissants : la technique des sacs de mots et l'indexation inversée. Ces outils permettent de calculer une signature pour chaque image du corpus et d'en fournir un accès rapide. Le module chargé d'interroger la base est donc à même de réaliser sa tâche efficacement.

La base logicielle a permis de tester différentes techniques de description locale sur un corpus d'images dédié aux tests. Les résultats obtenus donnent un bon aperçu des performances de chaque technique. Certaines techniques ont été mises en défaut, illustrant ainsi leurs limites et permettant de mieux cerner leurs capacités.

La base logicielle propose également de combiner plusieurs techniques de description et d'évaluer leurs complémentarités. Les spécificités et invariances des techniques figurant dans l'état de l'art ont ainsi été mises en avant lors des expériences. La combinaison semble être une voie prometteuse et doit encore être approfondie.

Ce stage m'a donc fait découvrir le domaine de la recherche et de l'indexation par contenu visuel. Appliqué au fond photographique du musée Nicéphore Niépce, j'ai pu constater l'intérêt pratique de ce domaine et me confronter à ses problématiques qui sont aujourd'hui de l'ordre de la recherche. Elles sont dûes en grande partie à la taille des données à traiter en temps réel et de nombreuses équipes travaillent sur le sujet.

En dehors de ces considérations techniques et théoriques, ces six mois m'ont également enrichi personnellement. Au contact de doctorants passionnés, j'ai pris plaisir à échanger et discuter sur nos domaines respectifs. La richesse technologique et humaine de la pépinière d'entreprise de Nicéphore Cité a suscité et aiguisé ma curiosité.

Je tire donc un bilan positif de ce stage et espère réussir mettre à profit ses enseignements au cours des trois années de doctorat que je m'apprête à commencer.

Bibliographie

États de l'art

[LA08] Jing Li and Nigel M. Allinson. A comprehensive review of current local features for computer vision. *Neurocomput.*, 71 :1771–1787, June 2008.

[MTS+05] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *Int. J. Comput. Vision*, 65 :43–72, November 2005.

[TM08] Tinne Tuytelaars and Krystian Mikolajczyk. Local invariant feature detectors : a survey. *Found. Trends. Comput. Graph. Vis.*, 3 :177–280, July 2008.

Détecteurs

[For07] Per-Erik Forssén. Maximally stable colour regions for recognition and matching. pages 1–8, June 2007.

[Hei04] Gunther Heidemann. Focus-of-attention from local color symmetries. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26 :817–830, July 2004.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151. The Plessey Company pic., 1988.

[KZB04] Timor Kadir, Andrew Zisserman, and Michael Brady. An affine invariant salient region detector. *Image Rochester NY*, 3021 :228–241, 2004.

[MCUP02] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *In British Machine Vision Conference*, volume 1, 2002.

[MGD98] P Montesinos, V Gouet, and R Deriche. Differential invariants for color images. In *Proceedings of the 14th International Conference on Pattern Recognition-Volume 1 - Volume 1*, ICPR '98, pages 838–, Washington, DC, USA, 1998. IEEE Computer Society.

[Mor77] Hans P. Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 2*, pages 584–584, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[MS04] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *Int. J. Comput. Vision*, 60 :63–86, October 2004.

[Sch96] Cordelia Schmid. *Appariement d'images par invariants locaux de niveaux de gris*. PhD thesis, Institut National Polytechnique de Grenoble, GRAVIR – IMAG – INRIA Rhône–Alpes, July 1996.

[TVG04] Tinne Tuytelaars and Luc Van Gool. Matching widely separated views based on affine invariant regions. *Int. J. Comput. Vision*, 59 :61–85, August 2004.

Descripteurs

[AHF06] Alaa E. Abdel-Hakim and Aly A. Farag. Csift : A sift descriptor with color invariant characteristics. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 1978–1983, Washington, DC, USA, 2006. IEEE Computer Society.

[BETVG08] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110 :346–359, June 2008.

[DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society.

[FA91] William T. Freeman and Edward H. Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13 :891–906, September 1991.

[Low99] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60 :91–110, November 2004.

[MS05] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27 :1615–1630, October 2005.

[MY09] Jean-Michel Morel and Guoshen Yu. Asift : A new framework for fully affine invariant image comparison. *SIAM J. Img. Sci.*, 2 :438–469, April 2009.

[SF96] Eero P. Simoncelli and Hany Farid. Steerable wedge filters for local orientation analysis. *IEEE Trans. Image Processing*, 5 :1377–1382, September 1996.

Sac de mots

[SZ03] Josef Sivic and Andrew Zisserman. Video google : A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1470–, Washington, DC, USA, 2003. IEEE Computer Society.

Indexation inversée

[C75] Alfonso F. Cárdenas. Analysis and performance of inverted data base structures. *Commun. ACM*, 18 :253–263, May 1975.

[ZMR98] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Trans. Database Syst.*, 23 :453–490, December 1998.

Glossaire

ACP
Analyse en Composantes Principales.
Technique de décorrélation qui permet de réduire l'information en conservant que les composantes principales. 22

Bash
Bourn-Again SHell.
Interpréteur en ligne de commande de type script (`http://www.gnu.org/software/bash/`). 47, 57

coin
Le terme de coin désigne un changement soudain d'une caractéristique au voisinage d'un pixel. Il peut donc s'agir d'un coin au sens commun du terme, mais également d'un point isolé, d'un bord à forte courbure, d'un changement de couleurs, etc. 11, 64

descripteur
Un descripteur a pour but de décrire les points d'intérêt obtenus par un détecteur. Il analyse le voisinage proche de chaque point, appellé support d'extraction, pour produire un vecteur caractéristique de cette zone. 10, 11, 14, 18, 19, 21–23, 27–29, 33–36, 38, 39, 41, 44–47, 49–57, 60, 61, 69, 70

DoG
Difference of Gaussians.
Une différence de gaussiennes est une technique utilisée par les descripteurs de type SIFT pour simuler les changements d'échelles. 14

détecteur
Un détecteur est un algorithmz qui permet d'isoler des zones d'intérêt dans une image. Ces zones d'intérêt peuvent être de deux types : des coins ou des régions. 10–19, 21, 23, 27–29, 33–35, 38, 39, 41, 44, 45, 47, 49–56, 60, 61, 64, 69, 70

EDI
Environnement de Développement Intégré. 31

entropie
Mesure du degré de désordre. 17

filtre de Prewitt et Sobel
Filtres qui permettent d'approximer les dérivées I_x et I_y d'une matrice A.

$$I_x = \begin{pmatrix} +1 & 0 & -1 \\ +k & 0 & -k \\ +1 & 0 & -1 \end{pmatrix} * A \qquad et \qquad I_y = \begin{pmatrix} +1 & +k & +1 \\ 0 & 0 & 0 \\ -1 & -k & -1 \end{pmatrix} * A$$

avec $k = 1$ pour Prewitt et $k = 2$ pour Sobel. $*$ est le produit de convolution. 12, 22

filtre Gaussien
Filtre qui s'applique à chaque terme (x, y) d'une matrice I de la façon suivante :

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma} \exp -(\frac{x^2 + y^2}{2\sigma^2})$$

Si I est une image, ce filtre a un effet de lissage et donc de réduction du bruit. Plus le paramètre σ est grand, plus le lissage est important. La gaussienne d'une matrice ou d'une image I s'écrit $G_\sigma(I)$. 12, 13, 15, 18, 66, 69

filtre Laplacien
Le filtre Laplacien d'une image est la dérivée d'ordre 2 à deux dimensions. En un point (x, y) d'une image, il vaut donc :

$$L(x, y) = \frac{\delta^2 I(x, y)}{\delta_x^2} + \frac{\delta^2 I(x, y)}{\delta_y^2}$$

Dans le domaine discret, les trois matrices suivantes permettent de calculer une bonne approximation du laplacien :

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}$$

. 44, 66

gnuplot
Outil de création de graphiques en ligne de commandes (`http://http://www.gnuplot.info/`). 31, 47, 50, 57

image intégrale
Image comportant en chaque pixel la somme de tous les pixels précédent vers la gauche et vers le haut. Elle se note $I_\Sigma(x)$ et est définit comme ceci :

$$I_\Sigma(x, y) = \sum_{i=0}^{i<x} \sum_{j=0}^{j<y} I(i, j)$$

Elle permet de calculer la somme des intensités d'une surface rectangulaire de l'image en seulement deux additions et une soustraction. 15, 21

ImageMagick
Outil de manipulation d'images en ligne de commandes (`http://http://www.imagemagick.org`). 31, 33

LoG

Laplacian of Gaussian.
Le laplacien de gaussienne d'une image I se note $L_\sigma(I)$ et vaut :

$$L_\sigma(I) = \sigma^2 * L(G_\sigma(I))$$

avec L(I) le laplacien et $G_\sigma(I)$ la gaussienne. 13, 20

mAP

Mean Average Precision.
La moyenne des précisions moyennes est calculée généralement sur une courbe de rappel/précision. Elle est la moyenne des valeurs de la précision, elles-mêmes moyennes des précisions de plusieurs requêtes pour un rappel donné. 50, 52, 54–56

matrice de Harris

Notée M, elle est le produit de convolution d'un filtre Gaussien avec la matrice des moments d'ordre deux d'une gaussienne de l'image I :

$$M_{\sigma_1,\sigma_2}(I) = \sigma_1^2 * G_{\sigma_2}(I) * A(G_{\sigma_1}(I))$$

où A est la matrice des moments d'ordre deux. 12, 13

matrice de Hesse

La matrice de Hesse, notée H, est le produit de convolution d'une gaussienne avec la matrice des dérivées partielles d'ordre deux d'une gaussienne de l'image I :

$$H_{\sigma_1,\sigma_2}(I) = \sigma_1^2 * G_{\sigma_2}(I) * B(G_{\sigma_1}(I))$$

. 13, 15

matrice des dérivées partielles d'ordre deux

Pour une image I donnée, la matrice des dérivées partielle d'ordre deux vaut :

$$B(I) = \begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{pmatrix}$$

. 15, 66

matrice des moments d'ordre deux

La matrice des moments d'ordre deux de I vaut :

$$A(I) = \begin{pmatrix} I_x^2 & I_x.I_y \\ I_x.I_y & I_y^2 \end{pmatrix}$$

. 16, 66

mesure cosine

Appellée aussi « similarité cosinus ».
Cette mesure permet de calculer la similarité entre deux vecteur de même dimension par la formule :

$$\theta = \arccos \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

Elle est comprise entre 0 et π, π étant la mesure de similarité maximale. Dans ce rapport, la mesure cosine sera uniquement la fraction. La similarité maximale sera donc 1. 27

NAS
Un NAS (Network Attached Storage) est un serveur de fichiers autonome relié à un réseau. 31

OpenMP
Open Multi-Processing.
Interface de programmation pour le calcul parrallèle (`http://www.openmp.org`). 31, 34, 39, 46

page-disque
Appellée aussi bloc.
C'est l'unité de lecture et d'écriture sur un disque dur. La taille d'une page-disque varie suivant le système de fichier utilisé et la taille de la partition. Il est par exemple de 4kio pour une partition NTFS de plus de 2Gio. 40

PGM
Portable GrayMap.
Format de fichier graphique en niveaux de gris. 43

point d'intérêt
Un point d'intérêt dans une image est une zone de pixels aux propriétés remarquables. Il s'agit le plus souvent d'une zone comportant des changements soudains d'intensité, mais il peut également s'agir d'une zone sans changements d'intensité ou encore avec des symétries locales. Les points d'intérêt peuvent prendre la forme de points, de courbes, ou encore de régions rectangulaires, circulaires, elliptiques.... 6, 8–11, 13–21, 27, 28, 30, 33–35, 37, 44, 45, 51, 53, 56, 61, 64, 68

produit de convolution
Le produit de convolution de deux fonction se définit ainsi :

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(x-t) \cdot g(t)\,dt = \int_{-\infty}^{+\infty} f(t) \cdot g(x-t)\,dt$$

Le produit de convolution de deux matrices de mêmes taille se réalise de la façon suivante :

$$\begin{pmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{pmatrix} * \begin{pmatrix} y_9 & y_8 & y_7 \\ y_6 & y_5 & y_4 \\ y_3 & y_2 & y_1 \end{pmatrix} = x_1 * y_1 + x_2 * y_2 \cdots + x_9 * y_9$$

Le produit de convolution en (x, y) d'une image I avec un noyau K est le produit de convolution de K centré en (x, y) avec la sous-matrice de I ainsi définie. 13, 15, 65, 66

précision
Il s'agit d'une mesure de performance d'interrogation de bases de données.
Elle est la proportion de bons documents retournés parmis l'ensemble des documents retournés. 41, 49–52, 60, 66

RAID 1
Redundant Array of Independent (or inexpensive) Disks 1.
Technique de duplication des données permettant d'écrire en parrallèle sur plusieurs disques dur. Chaque disque dur est une copie des autres. 31

RANSAC
RANdom SAmple Consensus.
Algorithme itératif d'estimation de paramètres. Permet d'estimer un modèle malgré la présence d'outliers. 21

rappel
Il s'agit d'une mesure de performance d'interrogation de bases de données.
Elle est la proportion de bons documents retournés parmis l'ensemble des documents à retrouver. 41, 49–52, 60, 66

région
Ensemble de pixels connexes répondant à un critère spécifique. 11, 64

signature
La signature d'une image est un vecteur représentatif de cette image. Ici, ce vecteur a une taille égale au nombre de mots du sac de mots. La i^{e} valeur de la signature correspond au nombre de vecteurs descripteurs dont le i^{e} mot est le plus proche. 9, 24–30, 32, 33, 37, 38, 54, 59, 60, 69

STL
Standard Template Library.
Il s'agit de la bibiothèque standard C++. 40, 41

tf-idf
Term Frequency-Inverse Document Frequency.
Mesure de l'importance d'un mot dans une image qui prend en compte non seulement le nombre d'occurence du mot dans l'image mais aussi dans l'ensemble des images de la base. 24–27, 59

vecteur descripteur
vecteur décrivant un point d'intérêt d'une image. Ne pas confondre avec le descripteur. 9, 10, 15, 18, 20–23, 29, 30, 32–36, 44–47, 49, 54, 58, 60, 68

Table des figures

Liste des algorithmes

ANNEXE A

Annexe du développeur et de l'utilisateur averti

A.1 Concevoir un module

A.1.1 Le fichier de configuration

L'utilisation de Qt simplifie énormément la construction d'un module. Tout est concentré dans un unique fichier d'extension *.pro* qui contient des déclarations de variables. Ce fichier permet entre autre de générer automatiquement le fichier Makefile par l'intemédiaire de l'exécutable qmake.s

Code source (A.1) – Exemple de fichier *.pro*

```
1 TEMPLATE = app
2 TARGET = detect
3 DESTDIR = ../bin
4
5 HEADERS += file1.h file2.h
6 SOURCES += main.cpp file1.cpp file2.cpp
7
8 INCLUDEPATH += libray_header_path
9 LIBS += -Llibrary_folder_path -lname_of_library
```

Pour réaliser un module, il convient donc de commencer par définir un fichier *.pro*. Dans le fichier .pro A.1, on peut constater que le contenu peut être très succinct. Les variables qui apparaissent ont les rôles suivants :

TEMPLATE : Définit s'il s'agit d'une application, d'une bibliothèque...
DESTDIR : Répertoire où sera créé l'exécutable/la bibliothèque
TARGET : Nom de l'exécutable
HEADERS et SOURCES : Liste des fichiers d'entêtes et les codes sources
INCLUDEPATH : Répertoires contenant les fichiers d'entêtes des bibliothèques tierces utilisées.
LIBS : Repertoires (-L) et instructions (-l) pour l'édition de liens des bibliothèques

Il ne s'agit bien entendu ici que d'un fichier de base, mais de nombreuses autres options existent. Vous trouverez plus d'informations sur la page de documentation de Qt : `http://doc.qt.nokia.com`.

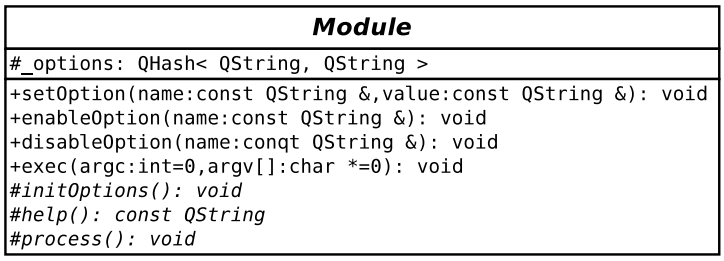


FIGURE A.1 – Diagramme UML de la classe abstraite `Module`

A.1.2 La classe `Module`

Pour devenir un module, il faut hériter de la classe abstraite `Module`, présentée sur la figure A.1. C'est cette classe qui gère les processus d'initialisation, d'option et d'exécution des modules.

Il y a trois fonctions abstraites à définir : `initOptions`(), `help`() et `process`(). Ces trois fonctions sont appellées par la fonction publique `exec`(), fonction principale qui exécute l'algorithm 4.

Algorithme 4 Algorithme de la fonction `exec`() de la classe `Module`

```
1: initOptions()
2: Si !hasArguments() or isEnableOption("help") alors
3:     print( help() )
4: sinon
5:     process()
6: Finsi
```

A.1.3 Les options

Les options sont resprésentées par un dictionnaire qui associe le nom de l'option à sa valeur sous forme de chaine de caractère. Une classe extérieure au module souhaitant connaitre la valeur d'une option devra donc connaitre au préalable son nom. C'est pourquoi il convient de définir les noms des options de chaque module en dehors de leur classe principale. Pour chaque module, les noms des options sont déclarées comme des chaines de caractères statiques avant la déclarartion de la classe. De plus, elles sont déclarées dans un espace de nommage dédié afin de permettre à deux module d'avoir une option de même nom.

L'initialisation des options doit être réalisée dans la fonction `initOptions`(). Il est essentiel d'initialiser toutes les options car l'utilisation d'une variable non initialisée déclenche une exception. Si elles sont obligatoires, il faut les initialiser avec la chaine de caractère vide.

A.1.4 L'exécution d'un module

Pour pouvoir exécuter un module, il est nécessaire de définir une fonction `main`(). Cette fonction doit ressembler au code source A.2 :

Code source (A.2) – Fonction `main`() type d'un module

```
1  int main(int argc, char *argv[])
2  {
3  try {
4      ModuleImpl module;
5      module.exec(argc,argv);
6  }
7  catch (QString e) {
```

```
 8       std::cerr << e.toStdString() << std::endl;
 9       return 1;
10   }
11
12   return 0;
13   }
```

Lors de l'utilisation d'un module en tant qu'exécutable, les options de la ligne de commande doivent être passées en argument de la fonction `exec()`. Cette fonction commence par modifier les valeurs des options et si l'option n'a pas été initialisée, une exception est déclenchée. Ensuite, si l'option `-help` fait partie des arguments, l'aide est affichée par l'appel à la fonction `help()`. Sinon, la fonction `process()` est exécutée.

A.1.5 Ajouter un algorithme de détection/description

Pour une première approche du fonctionnement des algorithmes de détection/description, vous pouvez vous référer à la section 3.4.2 page 33. Les classes auxquels va faire référence cette section sont visibles sur le diagramme de classes 3.4 page 35.

Prenons maintenant l'exemple de l'algorithme Plop, un nouvel algorithme de détection/description révolutionnaire dont les parties détection et description peuvent être éffectuées séparemment. Voici les étapes à suivre pour l'ajouter au module `Detector` :

Création La première étape est bien entendu de créer la classe Plop. La première chose à faire est d'hériter des interfaces `DetectorAlgorithm`, `DescriptorAlgorithm` et `DetectorDescriptorAlgorithm`. Si Plop avait été un détecteur, l'héritage de `DetectorAlgorithm` aurait suffit. On constate que ces trois interfaces comportent des fonctions abstraites de même profil. Il n'est necessaire de les redéfinir qu'une seule fois, ce qui permet entre autre de donner le même nom au détecteur et au descripteur. Le cœur de l'algorithme se situe respectivement dans les fonctions `extract()`, `describe()` et `extractAndDescribe()`.

Référencement Afin de rendre cet algorithme utilisable par le module `Matcher` et tout autre autre module qui en aurait besoin, il faut ajouter Plop au fichier `all_detectors.h`. Cela donnerait donc l'entrée « #include "algorithm/inc/plop.h" ».

Il est également important que chaque algorithme est un nom unique qui permette de l'instancier. Il est pour cela nécessaire de définir une constante dans le fichier `detectors_defines.h` contenant le nom de l'algorithme. C'est de préférence ce nom qui doit être utilisé comme valeur de retour de la fonction `name()` des interfaces. L'entrée pour l'algorithme Plop serait « #define PLOP_NAME "plop" ».

Enfin le module `Matcher` utilise le module `Detector` en tant que bibliothèque. Pour utiliser l'algorithme Plop, il faut donc l'intégrer à la bibliothèque. L'utilisation de Qt rend cette tâche facile, il suffit d'ajouter les classes de Plop au fichier *.pro* de la bibliothèque : `detector_lib.pro`. Les deux lignes à ajouter sont « HEADERS += plop.h » et « SOURCES += plop.cpp ». Au passage, si les deux fichiers `plop.h` et `plop.cpp` n'ont pas été créés via QtCreator, il est nécessaire d'ajouter aussi ces deux lignes au fichier `detector.pro`, sans quoi le module ne pourra être compilé.

Instanciation Afin de rendre l'algorithme instanciable, il est nécessaire de l'inscrire auprès des classes suffixées « Builder ». Pour un détecteur par exemple, il faudra l'inscrire auprès de la classe `DetectorBuilder`. L'algorithme Plop devra lui être inscrit auprès des trois classes `DetectorBuilder`, `DescriptorBuilder` et `DetectorDescriptorBuilder`. L'inscription est réalisée au sein des fonctions `constructMap()` de ces trois classes. Ces fonctions sont documentées et il est simple de calquer l'algorithme Plop sur les précédents. Notons que les algo-

rithmes sont ajoutés d'après leur nom définit dans le fichier `detectors_defines.h`, PLOP_NAME donc pour l'algorithme Plop.

Voilà, une fois ces étapes terminées, l'algorithme est prêt à être utilisé.

A.2 Les modules implémentés

A.2.1 Le module `Detector`

A.2.1.1 Options

Nom	Valeur par défaut	Description
-i		Dossier contenant la banque d'images.
-o		Nom du dossier de destination. Il sera amené à contenir également le sac de mots et les fichiers d'index.
-dd		Nom du détecteur/descripteur à utiliser pour l'extraction des points d'intérêts.
-det -desc		Noms du détecteur et du descripteur à utiliser pour l'extraction des points d'intérêt.

A.2.2 Le module `Classifier`

A.2.2.1 Options

Nom	Valeur par défaut	Description
-io		Nom du répertoire ou se trouve les fichiers produits par le module `Detector`.
-sub		Nom du sous-répertoire dans lequel écrire le fichier `words.list`. Cette options est facultative : le nom du sous-répertoire est alors assigné en fonction des options -k et -iter.
-algo	kmeans	Nom de l'algorithme de classification à utiliser Pour l'instant seul K-Means est implémenté.
-k	500	Nombre de clusters à créer (>1).
-iter	20	Nombre d'itération de l'algorithme de classification (>0).

A.2.3 Le module `Indexer`

A.2.3.1 Options

Nom	Valeur par défaut	Description
-io		Nom du répertoire créé par le module `Detector` et qui contient les fichiers produits par le module `Detector` et le module `classifier`.
-sub		Plusieurs classifications ont put être réalisées. Il s'agit donc du nom du répertoire contenant le fichier `words.list` désiré. Si cette option n'est pas spécifiée, le répertoire le plus récent est choisi.
-seq	false	Ajouter cette option pour obtenir un index séquentiel. L'index par défaut est un index inversé.

A.2.4 Le module `Matcher`

A.2.4.1 Options

Nom	Valeur par défaut	Description
-query		Nom de l'image dont on cherche les images semblables.
-i		Nom du répertoire créé par le module `Detector` qui contient l'ensemble des fichiers nécessaires.
-sub		Nom du sous-répertoire du répertoire spécifié par -i que l'on souhaite utiliser. Il contient le fichier `words.list` et le(s) répertoire(s) d'index. S'il n'est pas spécifié, c'est le sous-répertoire le plus récent qui est utilisé.
-idx		Nom du sous-répertoire du répertoire spécifié par -sub que l'on souhaite utiliser. Il contient les fichiers d'index produits par le module `indexer`. S'il n'est pas spécifié, c'est le sous-répertoire le plus récent qui est utilisé.
-o	images.same	Nom du fichier qui contiendra la liste des images similaires.
-dist	false	A spécifier uniquement si l'on souhaite obtenir toutes les images inférieures à une certaine distance. Par défaut, c'est un nombre d'images similaires à trouver qui est demandé.
-t	20	Seuil de recherche. Il s'agit d'une distance si l'option `-dist` est activée (entre 0 et 100) et d'un nombre d'images sinon (entre 1 et le nombre d'images indexées).
-elag	0	Permet de supprimer les mots qui apparaissent trop fréquement. Seules les mots qui ne figurent pas dans au moins -elag% d'images sont utilisés. On considère les autres comme non discriminants).

Vous pouvez également spécifiez un deuxième projet pour combiner les résulats. Les options sont -i2, -sub2, -idx2 et -elag2.

A.2.4.2 Algorithmes implémentés

Algorithme 5 Algorithme du calcul de la signature d'une image

```
1: VECTOR< DESCRIPTOR > descriptors
2: VECTOR< WORD > bag
3: SIGNATURE signature
4: FLOAT distanceMin, currentDistance
5: INTEGER winner, nbWordsFind
6: /* Compte pour chaque mot le nombre de vecteurs descripteurs dont il est le plus proche. */
7: Pour tout DESCRIPTOR D_i ∈ descriptors faire
8:    distanceMin = bag[0].distance(D_i)
9:    winner = 0
10:   Pour j = 1 → bag.size() − 1 faire
11:      currentDistance = bag[j].distance(D_i)
12:      Si currentDistance < distanceMin alors
13:         distanceMin = currentDistance
14:         winner = j
15:      Finsi
16:   Fin pour
17:   signature[winner] += 1
18: Fin pour
19: /* Calcul le nombre de 0 dans la signature. */
20: nbWordsFind = count( 0, signature )
21: nbWordsFind = signature.size() - nbWordsFind
22: /* Calcul du tf-idf pour chaque mot. */
23: Pour i = 0 → signature.size() faire
24:   signature[i] = (signature[i] / nbWordsFind)*bag[i].logNNi
25: Fin pour
```

Algorithme 6 Algorithme de chargement des signatures

```
1: VECTOR< INTEGER > wordsToLoad
2: MAP<INTEGER,SIGNATURE> index
3: STRING imageName
4: INTEGER nbImagesInWord, imageIdx
5: Pour i = 0 → wordsToLoad.size() − 1 faire
6:    FILE wordFile = open("wordi.idx")
7:    nbImagesInWord = wordFile.read()
8:    Pour j = 0 → nbImagesInWord faire
9:       imageIdx = wordFile.read()
10:      Si ! index.contains(imageIdx) alors
11:         index.insert(imageIdx,new Signature(wordsToLoad.size()))
12:      Finsi
13:      index[imageIdx][i] = wordFile.read()
14:   Fin pour
15:   wordFile.close()
16: Fin pour
```

A.2.5 Le module `PrecisionRecall`

Nom	Valeur par défaut	Description
-q		Nom de l'image requête.
-i		Nom du répertoire créé par le module `Detector` qui contient l'ensemble des fichiers nécessaires.
-sub		Nom du sous-répertoire du répertoire spécifié par -i que l'on souhaite utiliser. Il contient le fichier `words.list` et le(s) répertoire(s) d'index. S'il n'est pas spécifié, c'est le sous-répertoire le plus récent qui est utilisé.
-idx		Nom du sous-répertoire du répertoire spécifié par -sub que l'on souhaite utilier. Il contient les fichiers d'index produits par le module `indexer`. S'il n'est pas spécifié, c'est le sous-répertoire le plus récent qui est utilisé.
-elag	0	Permet de supprimer les mots qui apparaissent trop fréquement. Seules les mots qui ne figurent pas dans au moins -elag% d'images sont utilisés. On considère les autres comme non discriminants).
-o	PR_output	Repertoire de destination des graphes de rappel/précision.
- m	moyenne.pr	Nom du fichier pour le graphe moyen.

ANNEXE B

Sujet de stage

le cnam musée Nicéphore Niépce nicéphore cité Chalon sur Saône

Stage de recherche Master 2

CNAM Paris – Nicéphore Cité – Musée Nicéphore Niépce
2011

« Recherche par similarité visuelle fine dans les fonds photographiques numérisés »

Contexte

Dans le cadre d'une convention de recherche, l'équipe Vertigo du laboratoire CEDRIC (CNAM, Paris), le Pôle de développement Nicéphore Cité et le Musée Nicéphore Niépce (Chalon-sur-Saône), proposent un sujet de stage de recherche en master 2ème année, sur le thème de l'indexation et la recherche d'images par contenu visuel.

Vertigo[1] est une équipe de recherche en autres spécialisée dans les méthodes d'indexation par contenu visuel des grandes collections d'images et de vidéos. Elle étudie à la fois les méthodes de description du contenu visuel par analyse d'images, notamment les approches de description locale [HGBRM10, LTGBBB09, GBL08], et les méthodes d'accès rapides associées pour la gestion des grands volumes de contenus multimédia [GBBS09].

Nicéphore Cité[2] est un pôle de développement, de soutien et d'accompagnement de la filière image et son en Bourgogne. S'appuyant sur un large réseau de partenaires, la SEM Nicéphore Cité travaille sur l'élaboration et la conduite de projets d'entreprises innovantes, de recherche et de formations. Dans ce cadre, la SEM développe plusieurs niches stratégiques dont la Valorisation de contenu multimédia en étroite collaboration avec le Musée Nicéphore Niépce.

Le Musée Nicéphore Niépce[3], Musée de la photographie, a constitué en près de trente cinq ans d'existence l'une des collections photographiques les plus originales en Europe. Des premières héliographies de Nicéphore Niépce aux technologies numériques, cette collection de plus de trois millions d'images raconte les multiples histoires de la photographie dans son aventure esthétique et documentaire, mais aussi dans ses usages populaires et commerciaux.

Présentation du sujet

L'indexation et la recherche d'images par contenu visuel (CBIR pour « Content-Based Image Retrieval ») est une discipline de l'informatique qui a pour objectif la structuration automatique des grandes collections d'images en vue de pouvoir accéder efficacement à ces images selon des critères visuels choisis [GB06, DJLW08]. Les fonctionnalités offertes impactent de nombreux secteurs manipulant des ensembles d'images, comme l'audiovisuel, la culture, la sécurité, la santé, la recherche scientifique, etc. Nous souhaitons ici indexer par contenu visuel le fonds photographique du Musée de la photographie Nicéphore Niépce. Doté de plus de 3 millions de documents photographiques en cours de numérisation, leur indexation permettra une meilleure structuration de la collection numérisée et donc une manipulation plus facile et plus poussée, mais aussi contribuera à offrir de nouveaux scénarios d'interrogation au service de sa mise en valeur auprès du grand public.

---

[1] Vertigo : http://cedric.cnam.fr/vertigo/
[2] Nicéphore Cité : http://www.nicephorecite.com/
[3] Musée Nicéphore Niépce : http://www.museeniepce.com/

Indexer une image à partir de son contenu visuel nécessite d'abord de produire un résumé visuel de ce contenu, qui sera l'index de cette image dans la collection. On utilise pour cela des techniques d'analyse d'image pour extraire des caractéristiques visuelles qui décrivent son contenu pour une application donnée. Le fonds du Musée Nicéphore Niépce étant composé de documents photographiques aux contenus généralistes et éclectiques issus de supports divers, l'une de ses spécificités est la *redondance visuelle* qu'il peut recéler. Par exemple, deux collections acquises et numérisées peuvent contenir des photographies illustrant le même événement, montrant donc potentiellement des lieux, personnages ou scènes communs ; ou encore les pages d'une revue numérisée peuvent contenir des photographies par ailleurs déjà présentes dans le fonds photographique du Musée. Au vu des volumes mis en jeu, il devient de plus en plus difficile de repérer manuellement cette redondance ; pourtant, la traiter permet de mieux organiser et analyser le fonds photographique, par exemple simplement en éliminant les occurrences superflues, ou bien au contraire en mettant en relation les contenus ayant des points communs visuels, ou encore en étudiant les causes de cette redondance.

Parmi les approches de description du contenu visuel les plus performantes, on trouve les approches de *description locale* à base de points d'intérêt, qui sont efficaces pour la recherche de parties d'images ou la reconnaissance d'objets dans les images [TM08]. Ce type de solution est particulièrement approprié pour traiter le problème de la redondance visuelle du fonds photographique du Musée Nicéphore Niépce. Dans le cadre de ce stage, nous exploiterons les approches locales les plus évoluées actuellement, pour nous focaliser sur le cas de la recherche de correspondances d'images ou de parties d'images entre le fonds photographique et le fonds de revues numérisées du Musée. Ici, il existe plusieurs niveaux de différences entre photographies présentes dans une revue numérisée et photographies originales numérisées correspondantes : (a) photomontage de la photographie selon l'identité visuelle de la revue, ou artistique (recolorisation, troncage, déformation, etc.), (b) prises de vue associées à des moments ou à des points de vue différents (relevant par exemple de choix différents de photographies dans la planche-contact originale), ou plus difficile, (c) remake de la photographie originale respectant plus ou moins précisément son identité visuelle. Ces cas de figure impliquent des transformations photométriques, colorimétriques et géométriques entre images qui sont différentes dans chacun d'entre eux. Les approches de description locale classiques sont généralement invariantes ou robustes à un ensemble de transformations données, sans être capables de les isoler les unes des autres et de les contraindre, ou alors au prix d'une estimation a posteriori coûteuse. Elles ne seront donc pas à même de repérer et différencier efficacement – finement et rapidement – les cas de figure présents dans les fonds considérés. Après avoir étudié les approches de description locale de la littérature [TM08], le stagiaire devra proposer, développer et évaluer une approche de description unifiée et paramétrable selon le cas de figure (a), (b) ou (c) considéré. Appliqué au fonds photographique du Musée Nicéphore Niépce, ce sujet de recherche se situe au cœur du problème théorique général de la recherche de modèles de description des contenus visuels qui soient *discriminants* mais restent *robustes* face aux transformations que peuvent subir les images dans une collection au contenu hétérogène.

Organisation du stage

Il s'agit d'un stage de 6 mois à partir de mars 2011, rémunéré (750 euros net/mois) et basé principalement à Chalon-sur-Saône, avec des réunions de travail à Paris régulières. Selon les résultats obtenus par l'étudiant, il donnera lieu à une thèse de doctorat financée sur 3 ans, dans la continuité du sujet du stage, sur la structuration par contenu visuel et la mise en valeur du fonds photographique du Musée Nicéphore Niépce.

Compétences requises

De bonnes connaissances et une pratique en indexation d'images par contenu visuel, analyse d'images ou vision par ordinateur sont requises, ainsi que la maîtrise de la programmation C/C++ ou Java. Des connaissances élémentaires en bases de données (méthodes d'accès, index) renforceront le dossier de candidature.

Modalités de candidature

Envoyer par email aux deux encadrants :

- CV
- Lettre de motivation ciblée sur le sujet
- Plusieurs lettres de recommandation
- Relevé de notes des deux dernières années d'étude
- Liste des enseignements suivis et validés durant les deux dernières années d'étude

Encadrement et contacts

Encadrants :

- Valérie Gouet-Brunet, Maîtres de Conférences HDR CNAM – valerie.gouet@cnam.fr
- Gabriel Bloch, Directeur adjoint de Nicéphore Cité – gabriel.bloch@nicephorecite.com

Contact Musée Nicéphore Niépce :

Sylvain Besson, Responsable du service inventaire documentation au musée Nicéphore Niépce – Sylvain.BESSON@chalonsursaone.fr

Bibliographie

[DJLW08] R. Datta, D. Joshi, J. Li and J. Z. Wang, Image Retrieval: Ideas, Influences and Trends of the New Age, *ACM Computing Surveys*, 40(2): 1-60, 2008.

[GB06] V. Gouet-Brunet, Chapitre de livre « Recherche par contenu visuel dans les grandes collections d'images », *Encyclopédie de l'Informatique et des systèmes d'information*, pages 564–576, J. Akoka and I. Commyn-Wattiau (eds.), Vuibert, 2006.

[GBBS09] V. Gouet-Brunet, N. Bouteldja and M. Scholl, HiPeR : a hierarchical model for exact, approximate and progressive retrieval in multi-dimensional spaces, *International Journal on Data Engineering and Management*, 1(1) :14–33, 2009.

[GBL08] V. Gouet-Brunet and B. Lameyre, Object recognition and segmentation in videos by connecting heterogeneous visual features, *Computer Vision and Image Understanding (CVIU)*, 111(1) :86–109, February 2008.

[HGBRM10] N. Vu Hoang, V. Gouet-Brunet, M. Rukoz and M. Manouvrier, Embedding spatial information into image content description for scene retrieval, *Pattern Recognition Journal*, 43(9) :3013–3024, 2010.

[LTGBBB09] J. Law-To, V. Gouet-Brunet, O. Buisson and N. Boujemaa, ViCopT : a robust system for content based video copy detection in large databases, *ACM Multimedia Systems Journal*, 15(6) :337–353, December 2009.

[TM08] T. Tuytelaars and K. Mikolajczyk, Local invariant feature detectors: a Survey, *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008.