MASTER EN SCIENCES INFORMATIQUES

Microscopie virtuelle
localisation, extraction et classification des globules blancs dans un frottis sanguin

Peeters, Cédric

Award date:
2007

Awarding institution:
Universite de Namur

FUNDP
INSTITUT D'INFORMATIQUE
RUE GRANDGAGNAGE, 21
B-5000 NAMUR (BELGIUM)

UCL MONT-GODINNE

Microscopie virtuelle

Localisation, extraction et classification des globules blancs dans un frotti sanguin

Cédric Peeters

Promoteur : Jean-Paul LECLERCQ
Co-promoteur : Hubert MEURISSE

Année académique 2006-2007

Mémoire présenté en vue de l'obtention du grade de Maître en informatique

Résumé

Dans le cadre de la télémédecine, la télémicroscopie offre la possibilité de l'expertise à distance à partir de frottis numérisés et visualisables à l'aide d'un ordinateur. Ce mémoire présente les travaux effectués aux cliniques universitaires de Mont-Godinne pour le développement d'un microscope virtuel. De précédents travaux ont permis, à cette application de microscopie virtuelle, de numériser des frottis sanguins au format JPEG2000 et de visualiser ces frottis à l'aide d'un client de visualisation. Ce client offre les fonctionnalités intuitives d'un microscope conventionnel afin de permettre aux experts de le manipuler sans passer par une phase d'adaptation conséquente.

Le présent mémoire décrit une approche permettant de créer une galerie d'images à partir des frottis numériques, représentant les globules blancs. Il est aussi question de présenter cette galerie dans le client de visualisation, offrant ainsi la perspective de l'aide au diagnostic. La problématique abordée est celle de l'analyse automatisée des images à l'aide de divers traitements d'images et de la classification des globules blancs extraits. L'extraction automatique des globules se décompose en étapes, chacune utilisant un traitement d'image spécifique, parmi lesquels le seuillage, le filtrage, la détection de contours, la binarisation ou encore le chaînage. A l'aide des traitements d'images assimilés, un autre sujet traité est celui de l'amélioration de la qualité des images représentant les frottis numériques. Cette amélioration consiste à rendre l'image plus homogène permettant ainsi d'atténuer les variations de luminosité présentes.

Mots-clés : télémédecine, télémicroscopie, microscope virtuel, imagerie médicale, frotti numérique, traitement d'image, seuillage, filtrage, détection de contours, chaînage, homogénéisation, classification, JPEG2000.

Abstract

In the scope of telemedecine, telemicroscopy offers the possibility to do remote expertise from numerical smear, which we can visualize with the assistance of a computer. This master thesis presents the work carried out in the hospital of Mont-Godinne with the object of developing a virtual microscope. Previous works had already permitted this application to digitize blood smear using the JPEG2000 standard and to make it visible on a screen with a visualisation client. This client offers the functionalities of a conventional microscope in order to allow the experts to manipulate it without any adaptation phase.

This master thesis describes the approach which allowed us to create an images' gallery from a numerical smear, representing white blood cells. This gallery can be visualized whit the visualisation client which offers the possibility of automatically posing a diagnosis. The main problem concerns the automatic analysis of images using image processing. Another problem concerns the automatic white blood cells' classification. The automatic white blood cells' extraction is decomposed into different steps each of them uses a specific image processing amongst which: *thresholding*, *filtering*, *edge detection*, *binarisation* or *chaining*. Thanks to the image processing we can improve the quality of the numerical smear. The improvement consists in the attenuation of the luminosity variation accros the image.

Keywords : telemedecine, telemicroscopy, virtual microscope, medical imagery, numerical smear, image processing, thresholding, filtering, edge detection, chaining, homogenization, classification, JPEG2000.

Remerciements

Depuis le début de mon stage jusqu'à la remise de ce mémoire, presque 9 mois ce sont écoulés. Durant ce laps de temps, beaucoup de personnes ont collaboré à ce travail. C'est donc tout naturellement que je les remercie sincèrement.

Tout d'abord, je tiens à remercier mon promoteur, Jean-Paul Leclercq, Professeur à l'institut d'informatique à Namur. Sa bonne humeur et son enthousiasme ont été un plus à l'attention qu'il a portée à mon travail.

Bien entendu, ce stage n'aurait pas été possible sans Hubert Meurisse, mon maître de stage aux cliniques universitaires de Mont-Godinne. Le travail qu'il m'a proposé était très intéressant et l'était encore plus en voyant la passion dont il fait preuve pour le domaine médical. De plus, sa disponibilité pour réfléchir conjointement sur des problèmes est à souligner. Je le remercie pour tout ceci et également pour ses conseils avisés quant à la composition de ce mémoire et en sa qualité de co-promoteur.

Je suis aussi reconnaissant envers le Docteur Bernard Chatelain, chef de laboratoire, de m'avoir accueilli au laboratoire d'hématologie de Mont-Godinne et de s'être intéressé à mon travail, me donnant des conseils de perspectives de continuation pour celui-ci.

Je souhaite aussi remercier de tout cœur le personnel de l'hôpital dans lequel j'ai effectué mon stage, et particulièrement Yvan Cornet pour sa patience, son aide, ses explications, ses conseils et sa sympathie. Je n'oublie pas non plus Maïlys, stagiaire lui aussi à Mont-Godinne, pour l'humour et la bonne ambiance.

Les derniers, mais non des moindres, que je voudrais remercier de tout cœur sont mes parents et mon frère pour leur aide quant à la création et à la correction de ce mémoire. Un grand merci aussi à Melissa.

Table des matières

Table des figures

Introduction

Introduction

De nos jours, les télécommunications, qui désignent les communications à distance, sont un véritable phénomène de société et ce phénomène est mondial. Les technologies relatives à l'information ne cessent de progresser et surfent sur la vague de curiosité des individus, de leur soif d'informations et de la communication de ces dernières. Actuellement, la vie quotidienne est accompagnée d'images numériques, de sons numériques, de vidéos numériques, de téléphones portables, de connections à Internet avec ou sans fil, de paiements électroniques, de GPS, de courriers électroniques, et bien d'autres communications au format électronique. Au delà de la vie quotidienne, les télécommunications sont utilisées par bien d'autres domaines dont le domaine médical. Et plus particulièrement la télémédecine dont il est question dans ce travail. Effectivement, n'entend-on pas de plus en plus parler d'opérations chirurgicales menées à distance par un médecin situé à des kilomètres de là ? La télémédecine est l'art d'utiliser les TIC (Technologies de l'Information et de la Communication) afin d'améliorer les qualités des soins aux patients. On peut y inclure la téléintervention, la téléconsultation, la téléexpertise, la télépathologie et bien d'autres disciplines. Tout ceci permet d'améliorer l'efficacité médicale, de réduire les coûts, d'améliorer l'accessibilité aux soins, de réduire les déplacements, etc...

Plus précisément, ce travail concerne l'imagerie médicale et la microscopie à distance appelée la télémicroscopie (voir 1.3). Durant ma dernière année de maîtrise en informatique aux Facultés Universitaires Notre Dame de la Paix (FUNDP) à Namur, j'ai pu effectuer un stage de fin d'études aux cliniques universitaires UCL de Mont-Godinne. Ce stage s'est étalé du 11 septembre 2006 au 26 janvier 2007 et fait suite aux stages menés précédemment par d'autres étudiants des FUNDP et de l'IESN (Institut d'Enseignement Supérieur de Namur).

Ce mémoire présente les objectifs du stage, les recherches effectuées pour les atteindre, les résultats obtenus ainsi que des perspectives d'amélioration de ces résultats. Il se compose de quatre parties.

La **première partie** présentera le cadre du stage et ses objectifs. Elle commencera par présenter le laboratoire d'hématologie qui a servi de cadre au stage. Ensuite elle s'attardera sur une présentation de la télémicroscopie qui est le domaine concerné par ce travail. Après, elle continuera sur la présentation des divers travaux effectués par les stagiaires précédents qui consistent en l'élaboration d'un microscope virtuel. Ce microscope sert à l'exploitation de frottis[1] sanguins numérisés. Et enfin, cette partie se terminera par la présentation des objectifs du travail effectué durant le stage.

La **seconde partie** intitulée "matériel et méthode" s'emploiera à la présentation et

[1]Selon Le Petit Larousse Illustré, un frotti est une « *préparation en couche mince, sur une lame de verre, d'un liquide organique ou de cellules, prélevés en vue d'un examen microscopique* ».

l'explication des diverses techniques et matériaux utilisés lors du stage. Le thème principal de cette partie est le traitement d'image puisque le travail effectué concerne l'exploitation des images obtenues via le microscope présenté dans la partie précédente. Cette partie introduira quelques notions de base pour ensuite présenter le filtrage d'image et particulièrement quelques filtres utilisés lors du stage. Ensuite viendront les explications concernant la segmentation et le seuillage d'image. Puis, l'homogénéisation des images, qui consiste en l'amélioration des images, sera abordée. Et finalement, cette deuxième partie abordera les technologies permettant l'exploitation des images que sont DICOM et Telemis.

La **troisième partie** sera consacrée aux résultats obtenus par l'application, avec le langage Java, des méthodes décrites dans la seconde partie. Premièrement, elle approfondira ce qui constitua la part la plus conséquente du stage, c'est-à-dire la conception automatique d'une galerie d'images. Cette galerie contient des images de globules blancs détectés dans l'image d'un frotti sanguin obtenue à l'aide du microscope. Les différentes étapes, de la localisation des globules jusqu'à l'extraction de ces derniers, seront présentées. Ensuite cette partie traitera de l'homogénéisation des images, nommées méga-images et représentant un frotti sanguin, créées à l'aide du microscope et des travaux des stagiaires précédents. Finalement cette partie illustrera l'extraction d'une galerie d'images de globules contenues dans une base de données ainsi que la classification de ces images.

La **quatrième** et dernière partie reviendra sur la discussion des résultats obtenus. Elle abordera la corrélation entre les objectifs de départ et les résultats obtenus. Ensuite elle approfondira les limites de ces résultats et les améliorations qui pourraient être apportées. Et finalement diverses perspectives d'améliorations possibles du travail réalisé seront proposées.

Première Partie

Cadre et objectifs

1 Première partie : cadre et objectifs

1.1 Introduction

Cette première partie traite de la présentation du cadre dans lequel le travail, abordé par ce mémoire, a été traité. Pour ce faire, elle présentera le laboratoire d'hématologie des cliniques universitaires de Mont-Godinne (à Yvoir en Belgique). Elle passera ensuite à une description de la télémicroscopie, domaine qui nous concerne dans ce cas, puis décrira l'existant, qui constitue la base de départ. Cet existant consiste en un microscope virtuel, développé par de précédents stagiaires. Le cadre se compose donc du lieu de travail, du domaine (la télémicroscopie au service de l'hématologie) et de l'application déjà existante. Finalement, sur base de cet existant, les objectifs du travail seront décrits.

1.2 Le laboratoire d'hématologie

L'**hématologie** est une science médicale dont l'objet est l'étude du sang. Le sang est composé de globules rouges, de globules blancs et de plaquettes en suspension dans le plasma. Les **globules rouges**, aussi appelés érythrocytes ou hématies, sont l'élément majoritaire et contiennent l'hémoglobine. Ils permettent de distribuer l'oxygène dans le corps. Les **globules blancs**, aussi appelés leucocytes, servent au système immunitaire et sont divisés en diverses classes. Chacune de ces classes possède une fonction défensive propre. Les **plaquettes**, aussi appelées thrombocytes, permettent la coagulation du sang. Le **plasma** est un liquide biologique jaunâtre composé, entre autres, de sels minéraux, de protéines et d'eau.

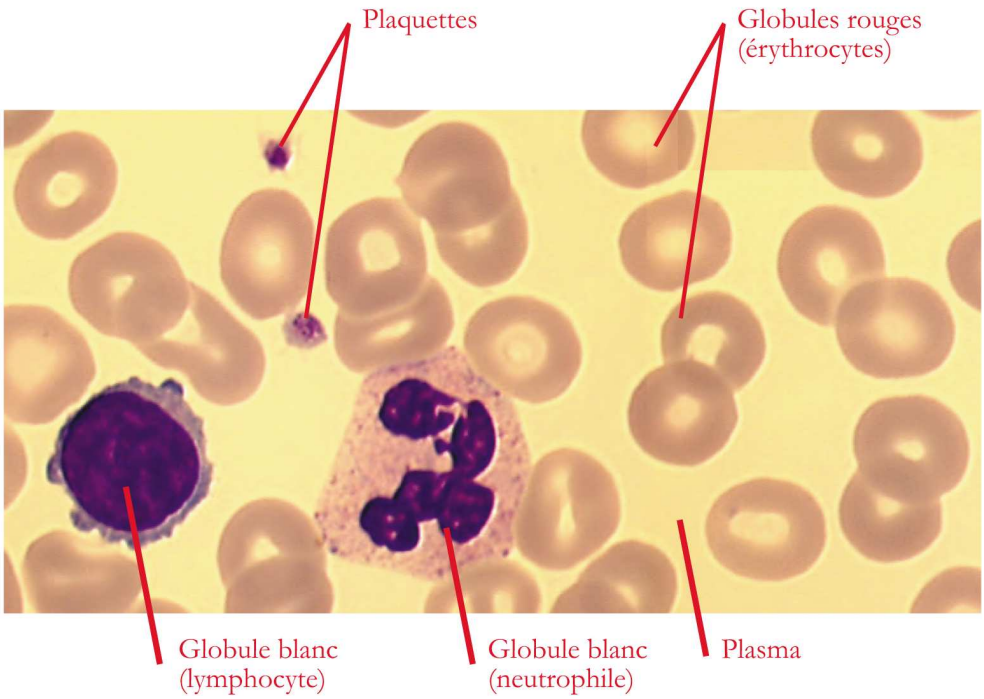

Fig. 1 – Sang vu au microscope

En hématologie, l'analyse **cytologique** est l'analyse, à l'aide du microscope, des éléments présents dans le sang. Il est donc question d'examiner la lignée érythrocytaire (les globules rouges), la lignée plaquettaire (les plaquettes) et les divers globules blancs.

Les **cliniques universitaires UCL**[2] **Mont-Godinne** possèdent un laboratoire d'hématologie qui traite des échantillons et fournit rapidement des résultats pour les examens dits "de routine", cela grâce à l'automation. Ce laboratoire s'occupe notamment des analyses cytologiques. C'est ce laboratoire qui a servi de cadre au stage.

En conclusion, le domaine abordé par ce travail est celui de l'examen du sang à l'aide d'un microscope.

1.3 La télémicroscopie

L'imagerie médicale est un domaine qui est de plus en plus adapté aux technologies de l'information et de la communication. L'intérêt pour la numérisation d'image, en provenance de caméras ou appareils photos, est croissant. L'objectif de cette adaptation aux TIC est un acheminement rapide des images à partir de leur point de production jusqu'aux différents postes de consultation par des spécialistes. De plus, ces technologies permettent aussi de faciliter l'interprétation et l'exploitation des images capturées. L'imagerie médicale permet d'examiner un patient via l'acquisition d'images internes du corps humain et se compose, par exemple, de l'endoscopie, l'échographie, l'image par résonance magnétique (IRM) ou encore la radiographie. L'application abordée dans ce travail est la microscopie et plus précisément la télémicroscopie.

La **télémicroscopie**[3] peut être définie comme étant l'application des TIC à la microscopie dans un but de téléexpertise, de télédiagnostic ou encore de télépathologie. Ceci implique que les images visibles via un microscope sont capturées par un ordinateur et observables sur ce dernier. L'expert peut ainsi examiner le frotti via un ordinateur pour émettre un diagnostic. Le microscope peut se situer à plusieurs kilomètres de l'ordinateur permettant la visualisation. Les fonctionnalités traditionnelles d'un microscope, telles que le zoom, la focalisation ou le déplacement sont accessibles via la télémicroscopie. A cela, elle ajoute des fonctionnalités liées à l'ordinateur qui permettent de faire du traitement d'image ou encore d'annoter les images de remarques ou de points de repères. Avec un tel système, le temps et la qualité de diagnostic s'en trouvent amélioré. En effet, il n'est plus nécessaire d'envoyer à un expert distant la lame de l'échantillon à examiner. Il suffit de lui envoyer, à l'aide des TIC, les images capturées. La connaissance devient décentralisée, les coûts diminuent, la productivité augmente, l'archivage est facilité. Bien entendu ceci est une vision dans le meilleur des cas et dans un monde qui ne prendrait pas en compte le facteur humain. En effet, l'ordinateur ne possède pas la connaissance de l'expert. De plus, il faut tenir compte des disponibilités de l'expert qui ne sera pas forcément disponible pour

[2] Université Catholique de Louvain La Neuve

[3] Pour plus de précisions sur la télémicroscopie, [Jad05] offre un état de l'art précis sur le sujet.

émettre un diagnostic dès que les images seront prêtes. De surcroît, le couple microscope-ordinateur n'est certainement pas infaillible et il nécessitera la supervision d'un acteur humain. Certains microscopes ne sont pas équipés de platines motorisées ou de changement d'objectif automatisé, ce qui nécessite la présence d'une personne sachant agir sur le microscope comme souhaité.

Les mémoires [Geo02], [Zuy03], [Jad05] de précédents stagiaires différencient deux types de systèmes de télémicroscopie. Dans [Geo02] il est question d'un système statique et d'un système dynamique. Ces deux types sont repris dans [Zuy03] et [Jad05] et nommés "store and forward" pour le système statique et "real time" pour le système dynamique.

L'approche dynamique appelée **real time** consiste en un système permettant de piloter à distance le microscope et d'en voir les effets quasiment immédiatement sur l'ordinateur permettant la visualisation et le pilotage. Le microscope et ses fonctionnalités sont contrôlables en temps réel. L'utilisateur, pilotant à distance le microscope, peut ainsi visualiser sur son moniteur ce qu'il pourrait observer sur place au travers du microscope. Il est donc libre de visualiser ce qu'il souhaite de façon à analyser le frotti en direct. Le désavantage est qu'il faudra remettre la lame en place sous le microscope dès que des analyses ultérieures devront être faites.

La seconde approche appelée **store and forward** correspond à un système de télémicroscopie où l'acquisition des images et la visualisation de celles-ci ne se font pas de façon synchrone. Ce système commence par la numérisation des images visibles via le microscope (souvent une portion significative d'un frotti, déterminée par une personne experte en la matière, plutôt que son entièreté). Les images numérisées sont stockées et peuvent ensuite être visualisées sur place ou transmises. Ces images seront disponibles pour de futures analyses sans devoir refaire intervenir le microscope. De plus ces images resteront inchangées au cours du temps, là où un frotti sur une lame va évoluer et s'altérer. Pour pallier au fait qu'il n'est plus possible d'utiliser les fonctionnalités du microscope en tant que telles, les frottis peuvent être numérisés à divers grossissement à l'aide des divers objectifs oculaires du microscope. De ce fait, il n'y aura pas une seule image pour une lame mais plusieurs images, représentant chacune des points de vue différents.

1.4 Le microscope virtuel

Le travail abordé dans ce mémoire s'inspire de l'approche store and forward de la télémicroscopie. Un appareillage de capture digitale couplé à l'optique d'un microscope robotisé permet en effet de générer automatiquement des observations numériques. Les frottis numérisés sont stockés sur un ordinateur et les images concernées peuvent être analysées quand bon nous semble. Aux cliniques universitaires de Mont-Godinne, l'ensemble de ce système, permettant d'acquérir et de visualiser des images, est appelé **microscope virtuel**.

1.4.1 *Acquisition d'images*

L'acquisition nécessite obligatoirement un microscope. Celui utilisé au laboratoire est un **microscope Olympus AX70**. Ce microscope possède une platine motorisée permettant de déplacer la lame à observer sur les axes X, Y et Z. L'axe X permet de se déplacer de gauche à droite, l'axe Y permet de se déplacer d'avant en arrière et enfin, l'axe Z permet de se déplacer de haut en bas. De plus il possède également un changement d'objectif motorisé. Les objectifs présents sur cet appareil sont des objectifs de grossissement 10x, 20x, 40x et 100x. Ce microscope est accompagné d'un boîtier permettant l'autofocus (boîtier U-AF d'Olympus). L'autofocus déplace, de façon automatisée, la platine de haut en bas, donc selon l'axe Z, jusqu'à l'obtention de l'image la plus nette. Ce boîtier U-AF est accompagné d'un boîtier de commande manuelle permettant de se déplacer sur l'axe Z de façon manuelle. Ce déplacement peut se faire en mode rapide avec une précision de 500 μm (position "coarse" du bouton de mise au point) ou de façon plus précise (position "fine" du bouton de mise au point) avec une précision de 300 μm pour l'objectif 10x, de 100 μm pour l'objectif 20x, de 30 μm pour l'objectif 40x et de 5 μm pour l'objectif 100x. Un autre boîtier, nommé U-MCB (Unit-Multi Control Board), se trouve relié au microscope et permet, par exemple, le réglage de la luminosité, des couleurs ou encore des objectifs. Ce microscope est contrôlable à distance via des commandes respectant le standard RS232[4] à l'aide d'une connexion par port série. Il est possible via ces commandes d'effectuer, à distance, des réglages de mise au point comme le déplacement des platines. Une caméra SONY DXC950 est installée sur ce microscope afin de pouvoir visualiser, sur un écran d'ordinateur, ce qui peut être observé au travers des objectifs oculaires. Cette caméra permet la capture d'images. L'illustration de la figure 3 nous montre le microscope relié à un ordinateur utilisant AnalySIS Pro 3.0. La figure 2 montre le boîtier de commande manuelle (non visible sur la photo de la figure 3).

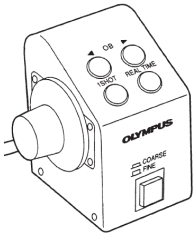

FIG. 2 – Boîtier de commande manuelle

[4]Standard permettant l'interconnection d'appareils provenant de différents fabriquants. Ce standard assure la fiabilité des communications et spécifie le voltage des signaux, le timing de ceux-ci et leur fonction. Il spécifie aussi un protocole d'échange d'informations.

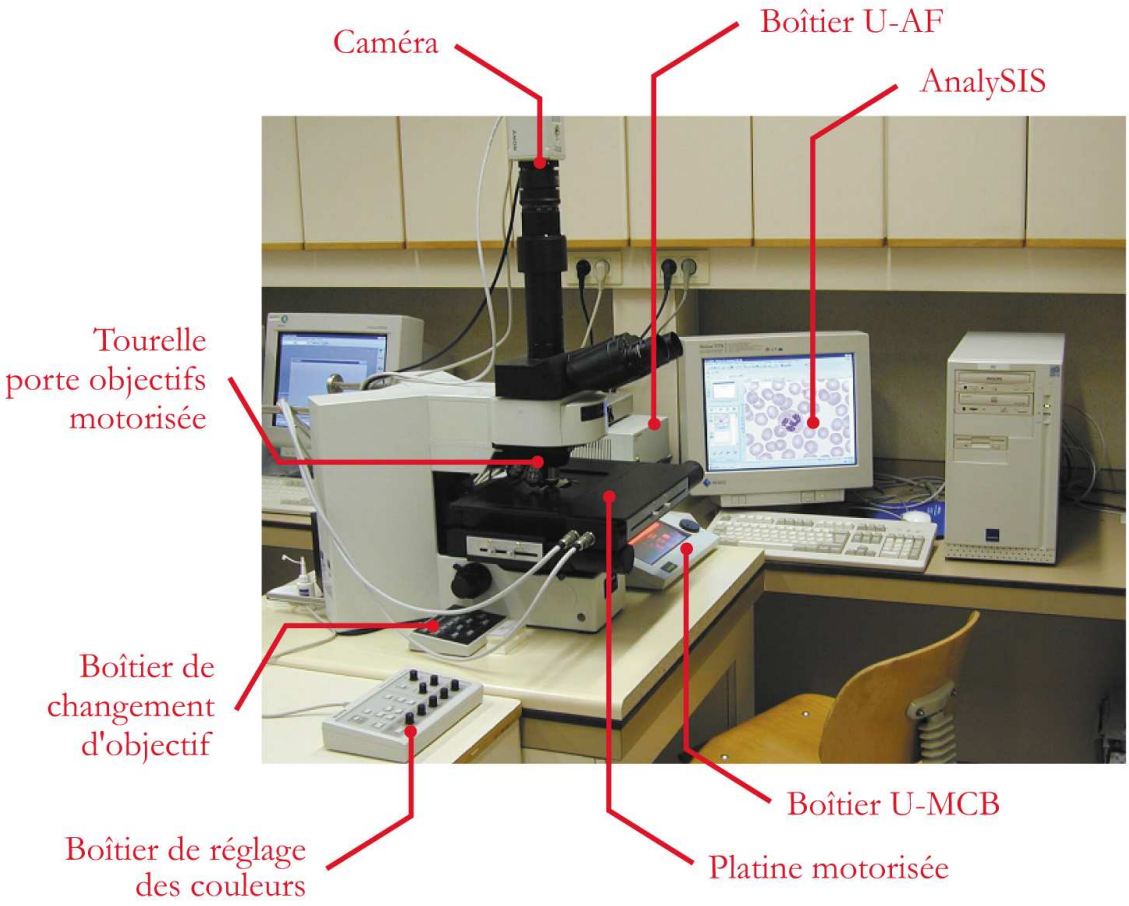

FIG. 3 – Station d'acquisition

Cet ensemble d'appareils composant le microscope est relié à un ordinateur équipé du logiciel **AnalySIS Pro 3.0**[5] (de Olympus Soft Imaging Solutions gmbh) fournissant la possibilité de contrôler à distance le microscope via le module « AX70 ». Ce logiciel permet l'acquisition et le traitement d'images. Ce logiciel possède aussi l'avantage d'être personnalisable via la création de modules à l'aide du langage Imaging-C compatible avec le standard ANSI-C. Ce langage fournit des fonctions d'analyse d'image (un simple exemple est la fonction *bmpCountColors*() qui permet de comptabiliser le nombre de valeurs de pixel différentes dans une image). L'environnement d'Imaging-C possède un éditeur de code, un compilateur, un interpréteur et un « debugger ».

Ce couple microscope-ordinateur permet l'acquisition d'images. Cette acquisition peut se faire de façon manuelle mais aussi de façon automatique. Les travaux effectués par B. Georges (voir [Geo02]) ont permis, entre autres, l'acquisition automatique d'une **galerie d'images des régions d'intérêt** d'une zone d'un frotti. Pour ce faire, un premier balayage de la zone se fait avec l'objectif de grossissement 20x afin de repérer des régions

[5]Un visuel du logiciel se trouve à la figure 51 en annexe B

d'intérêt. Ensuite l'objectif de grossissement 100x est utilisé afin de numériser les régions d'intérêt déterminées précédemment et d'en composer une galerie d'images. Cette galerie est encodée au format TIFF et l'outil permettant de la créer est intégré au logiciel AnalySIS.

Ces travaux posent des bases pour ceux effectués par L. Zuyderhoff (voir [Zuy03]) notamment pour le parcours d'une zone d'un frotti, pour la capture d'images, le changement d'objectif ou encore la gestion des buffers. Les travaux de L. Zuyderhoff introduisent le concept de **méga-image** et vont permettre l'acquisition de telles images. Des méga-images sont des « *images dont les dimensions sont gigantesques, c'est-à-dire, des images qui nécessitent une grande quantité de ressources pour leur stockage et qui entraînent des délais d'accès importants* »[Zuy03]. L'acquisition de méga-images permet ainsi de numériser une partie[6] de frotti numérique. Le microscope ne permettant que la capture image par image, les méga-images sont donc composées à partir de plusieurs captures (une capture est une image de 768 pixels × 573 pixels) qu'il faut aligner correctement. Cette procédure d'alignement de plusieurs captures, avec recouvrement partiel de celles-ci, en une grande image est appelée coregistration. Voici ci-dessous un exemple de coregistration ratée (à gauche) et de coregistration réussie (à droite) entre deux captures d'un frotti sanguin.

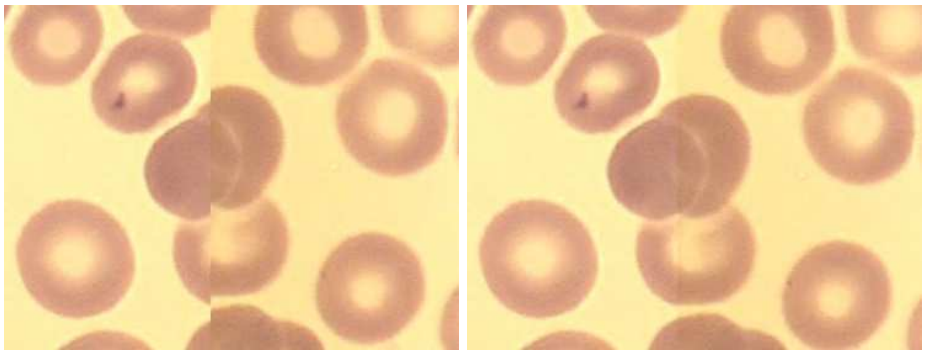

FIG. 4 – Exemple de coregistration ratée et réussie

Cette acquisition demande de définir la taille (largeur × hauteur)de la méga-image en nombre de captures. Les images sont capturées par un **client de création** et coregistrées par un **serveur de création** aussi appelé serveur de coregistration. Ce dernier compose les méga-images à partir des images capturées. Au final, une méga-image est une matrice d'images et est encodée au format **JPEG2000**[7]. JPEG2000 est un format d'image utilisant la compression par ondelettes (technique du Discrete Wavelet Transform). C'est donc ce format qui servira au stockage des frottis numérisés. La figure 5 est l'illustration de quatre images composantes et de la "petite" méga-image qui en résulte :

[6]Malgré sa grande taille, une méga-image représentera seulement une partie de frotti (idéalement, 1 cm^2). Effectivement, avec un objectif de grossissement 100x, la taille d'une capture visible à l'écran via la caméra du microscope est de 61,72 μm de large sur 46,21 μm de haut et est composée de 768 pixels de large sur 573 pixels de haut. Or la taille d'un frotti peut se mesurer en centimètres. Numériser un frotti entier serait trop coûteux en temps et en espace de stockage.

[7]Ce standard est largement décrit dans [Zuy03] et [Jad05]

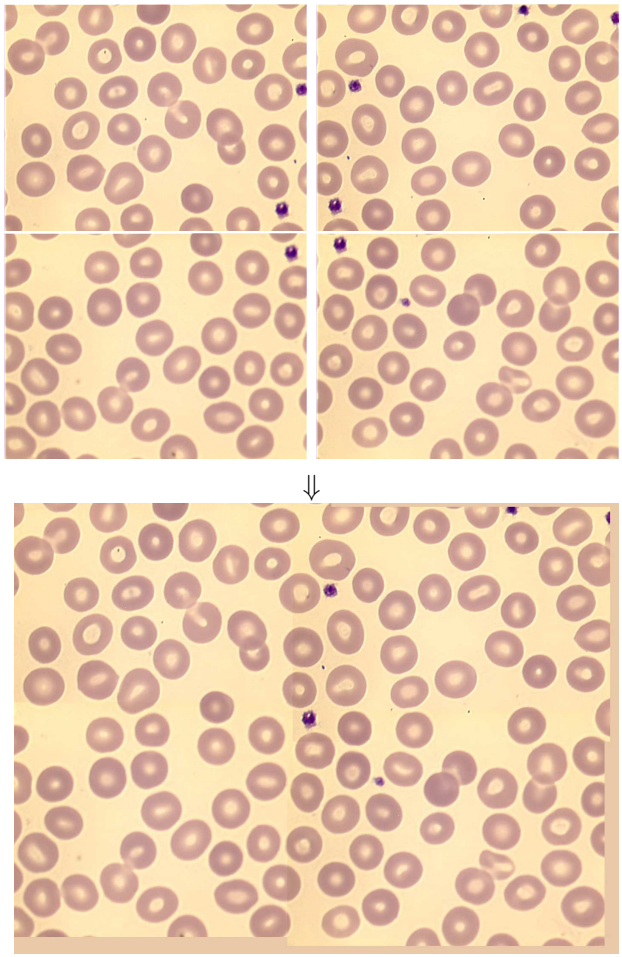

FIG. 5 – Coregistration : exemple de méga-image, matrice de 4 images

Suite à cela, G. Decocq, dans ses travaux (voir [Dec04]) propose d'améliorer la coregistration d'images. En effet, les méga-images obtenues sont encore trop petites à cause de la nécessité de charger chaque image la composant en mémoire. L'idée développée dans ces travaux était d'effectuer la coregistration en ne chargeant que les bords des images composantes. C'est finalement L. Zuyderhoff qui a amélioré à nouveau la coregistration en l'appliquant image par image, ce qui ne nécessitait plus de charger toutes les images en mémoire. De plus, un système de cache est mis en place permettant de gérer les images composantes déjà chargées en mémoire.

1.4.2 *Visualisation d'images*

Une fois les images numérisées d'une zone d'un frotti sanguin acquises et stockées, il faut pouvoir les visualiser et les exploiter.

A nouveau, les travaux de L. Zuyderhoff s'attaquent à ce problème. En effet, il pro-

pose une première mouture d'une **application de visualisation** permettant d'afficher une méga-image. Cette application se présente sous la forme d'un client et d'un serveur répondant aux requêtes du client. Le **client de visualisation** permet l'affichage d'une méga-image stockée au niveau du serveur et propose quelques fonctionnalités de zoom, de filtrage de couleur et de déplacement selon les axes X et Y dans le frotti numérisé. Le zoom permet d'afficher le frotti selon différents grossissements. Le filtrage de couleur permet de modifier les couleurs des frottis numérisés en jouant sur les trois composantes du codage RGB (voir 2.2) des pixels de ces derniers. Un filtre dénommé "lightness" permet de donner un effet de modification de la luminosité. Enfin, le déplacement permet de se déplacer de gauche à droite et de bas en haut dans le frotti numérisé. En effet, il se peut qu'un tel frotti numérisé soit plus grand que l'espace d'affichage du client de visualisation, il est donc important de pouvoir se déplacer dans cette image pour pouvoir visualiser les parties souhaitées du frotti. Le **serveur de visualisation**, qui a accès directement aux méga-images stockées, s'occupe de satisfaire les requêtes du client en lui fournissant les parties désirées d'une méga-image à la résolution souhaitée. Ce serveur est équipé d'un système de cache pour améliorer l'efficacité de rapatriement des parties d'image par le client.

Par la suite, les travaux de C. Jadoul (voir [Jad05]) améliorent grandement le visualisateur de méga-images. Ce visualisateur fonctionne toujours selon une architecture client-serveur et son avantage par rapport au précédent visualisateur se matérialise par une navigation dans l'image nettement plus efficace. Cette navigation efface les problèmes de lenteur via une gestion de la mémoire optimisée par un **système de cache et de prefetching**. Ce prefetching consiste à charger des parties d'images à l'avance. Ces parties d'images sont calculées par un algorithme et représentent les données les plus susceptibles d'être demandées dans un futur proche par le client (typiquement les parties directement adjacentes à la partie visible à l'écran via l'application de visualisation). La rapidité de rapatriement des parties d'images du serveur vers le client a été accrue en permettant au client de décompresser les données obtenues via une requête. En effet, cette solution permet au serveur de ne pas décompresser les données avant de les envoyer, ce qui améliore son temps de réponse (principalement dans le cas où plusieurs clients le sollicitent). De plus, le volume des données envoyées est réduit puisqu'elles ne sont pas décompressées, le transfert de ces dernières gagne donc en rapidité aussi. Cette application offre aussi une amélioration des fonctionnalités de l'application de visualisation. Il est maintenant possible de se déplacer dans l'image, en plus du système de navigation déjà existant, simplement en cliquant sur la zone de l'image à visualiser. Le choix de cette zone se fait dans une vision d'ensemble de la méga-image en version miniaturisée appelée "slide view". De plus, la navigation peut voir sa vitesse changer à l'aide d'un curseur définissant la vitesse souhaitée. Ainsi, en plaçant le curseur tout à fait à droite, le défilement de l'image est à sa vitesse maximum. Et inversement, le curseur positionné à l'extrême gauche permettra de ralentir le défilement de l'image. Il est aussi possible maintenant de choisir directement le grossissement souhaité sans passer par les divers grossissements intermédiaires. A ces améliorations s'ajoutent de nouveaux outils tels que la **capture d'images** ou encore le **comptage des cellules**. La capture d'images permet d'enregistrer une image au format JPEG en sélectionnant la zone à capturer. Le comptage des cellules est un outil qui permet

de comptabiliser manuellement les globules blancs selon leur classe.

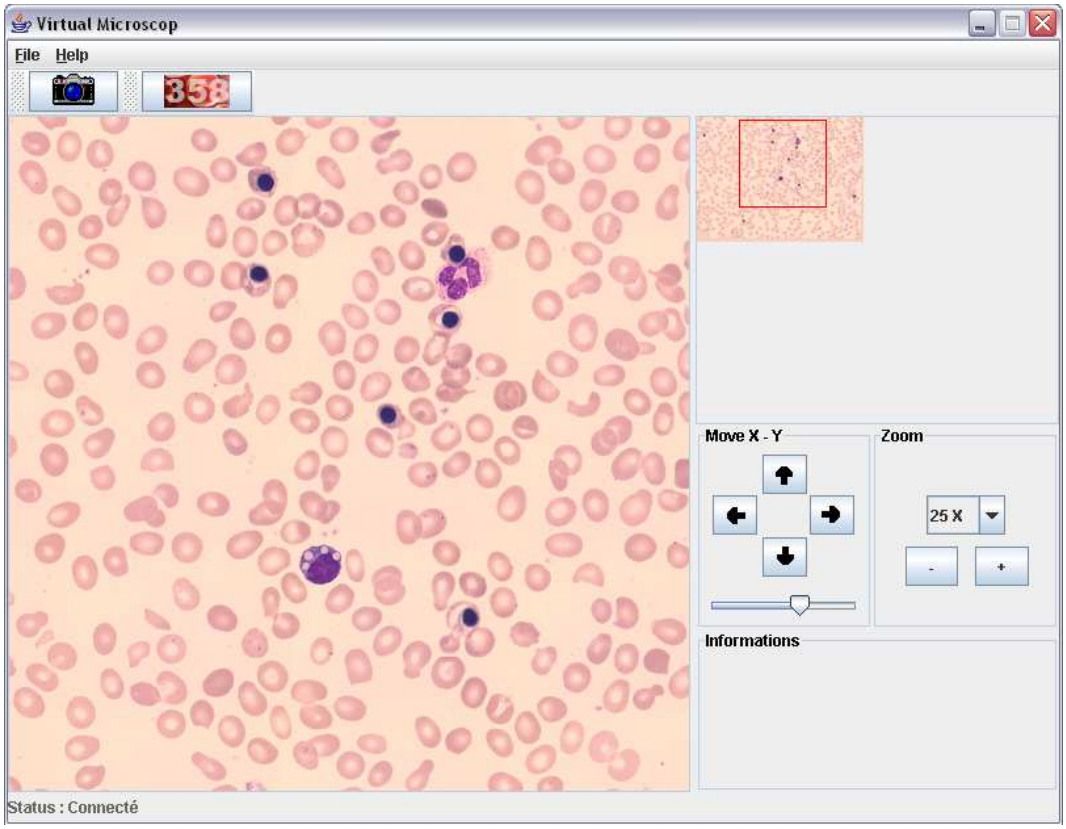

FIG. 6 – Application de visualisation

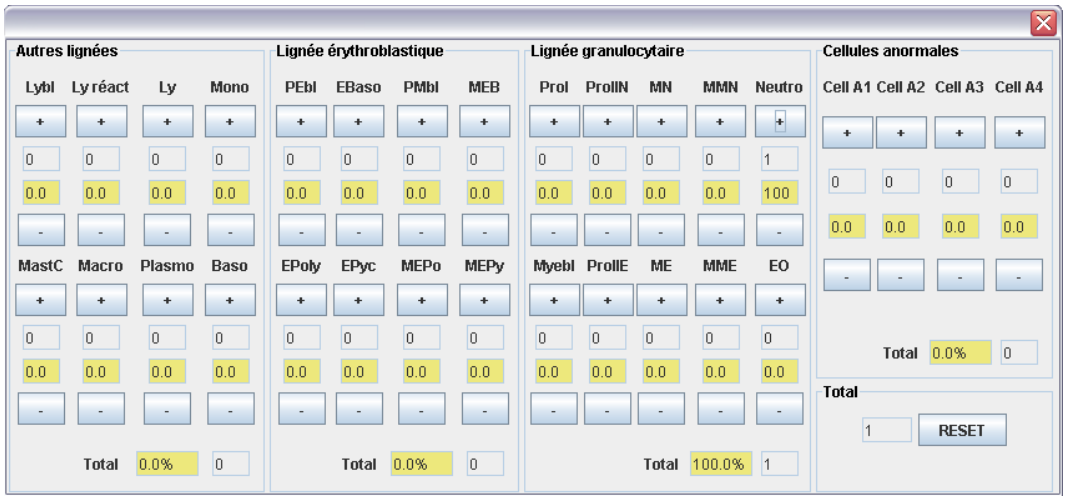

FIG. 7 – Outil de comptage

1.4.3 *Une architecture client-serveur*

Comme décrit précédemment, le microscope virtuel développé à Mont-Godinne se compose de l'acquisition et de la visualisation. Chacune de ces deux subdivisions est basée sur une architecture client-serveur comme le montre le schéma ci-dessous.

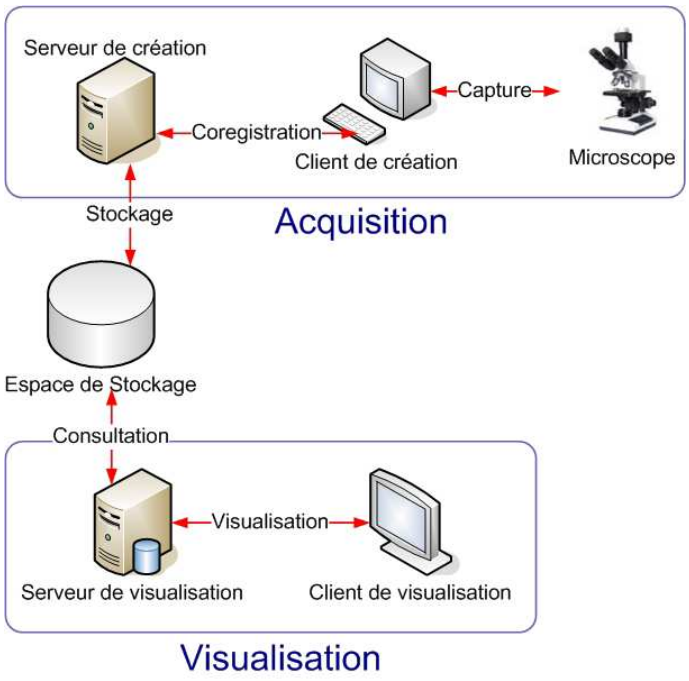

FIG. 8 – Microscope virtuel : architecture client-serveur

Concernant la partie d'acquisition (composée du **client de création** et du **serveur de création**, dédiés à la création des méga-images), l'architecture de client-serveur se justifie par le fait que le client de création, initiant la création d'images, peut se trouver sur une machine peu performante et possédant des ressources limitées. Le processus coûteux de coregistration peut ainsi se dérouler sur un serveur distant beaucoup plus adapté à ce type de processus. De plus, le stockage des images nécessite un espace de stockage suffisant, ce qui n'est pas forcément le cas de la machine cliente. Ainsi, le client s'occupe d'acquérir les images une par une et le serveur les réceptionne, les assemble correctement et stocke le résultat final.

En ce qui concerne la partie de visualisation, une telle architecture devient indispensable en ce sens que la visualisation est censée se faire sur une multitude de postes distants. Ces postes distants présentent une grande variété de configurations étant donné que les personnes visualisant les méga-images peuvent être géographiquement réparties (il est donc peut probable qu'elles possèdent toutes un ordinateur d'une même configuration). Il est donc nécessaire que l'application de visualisation puisse fonctionner sur une grande variété de configurations. La découpe client-serveur permet de tenir compte de cette diversité

d'environnements. En effet, le stockage des méga-images et l'accès à ces dernières ou à des parties de ces dernières se fait ainsi sur un **serveur de visualisation** possédant les capacités nécessaires à ces activités. Le **client de visualisation** s'occupe finalement d'afficher les parties de méga-image envoyées par le serveur. Les travaux de C. Jadoul offrent un protocole de communication entre le client de visualisation et le serveur de visualisation (voir [Jad05]). Ce protocole se présente sous la forme d'un code suivi de paramètres. Le code identifie quatre classes de requêtes/réponses : la gestion des connections, la gestion des informations des méga-images, la gestion de l'envoi des parties de méga-images et la gestion des erreurs. Bien entendu, il est possible de connecter plusieurs clients de visualisation à un même serveur de visualisation.

Le schéma de la figure 8 montre un espace de stockage séparé des deux serveurs. Effectivement, cet espace de stockage peut être disjoint des deux serveurs, tout comme il peut être présent sur un des deux serveurs, voire sur les deux (dans le cas où les deux parties serveurs sont localisées sur une même station par exemple).

1.5 Objectifs du mémoire

Ce mémoire, en continuité avec celui de C. Jadoul, ouvre des perspectives dans le cadre du diagnostic et de l'expertise à distance pour ce système de télémicroscopie dédié aux analyses cyto-hématologiques.

Divers objectifs ont été fixés dans un but d'amélioration du système actuel. La première chose à faire était de prendre connaissance des divers ouvrages écrits par mes prédécesseurs. Bien entendu, un objectif inhérent à ces lectures consistait à faire fonctionner ce système correctement et à en saisir le fonctionnement et le paramétrage. Il était aussi question de comprendre les technologies utilisées pour ce système. Nous avons dégagé quatre objectifs principaux dont un qui est apparu en cours de route et un autre qui a dû être abandonné, nous verrons pourquoi.

A partir du microscope virtuel en l'état décrit précédemment, le premier objectif est de pouvoir **recréer une galerie d'images à partir d'une méga-image**. En effet, la création d'une galerie des régions d'intérêt proposée par B. Georges a été remplacée par l'acquisition de méga-images. Il est donc intéressant de pouvoir obtenir une galerie des régions d'intérêt de façon automatique à partir des méga-images. En l'occurrence, les régions d'intérêt concernées sont les globules blancs d'un frotti sanguin. Bien sûr, ce premier objectif se décomposait en divers sous-objectifs, à commencer par trouver une idée pour l'atteindre, puis faire des recherches pour mettre en oeuvre ces idées. Ces recherches portaient notamment sur le traitement d'image, discipline quasiment inconnue pour moi, et son application à l'aide du langage Java utilisé pour créer le système. Il a aussi fallu passer par une phase de tests de traitements d'image via le logiciel ImageJ. De plus, pour pouvoir modifier le microscope virtuel de façon à atteindre cet objectif, il fallait prendre le temps de comprendre le code source de ce dernier afin de savoir ce qu'il fallait modifier et ajouter.

Certains pourront penser que la création de la galerie d'images pouvait se baser sur celle créée par B. Georges. Cependant je n'ai jamais vu cette application, ni même son code source. Je n'ai pas non plus cherché à voir cette dernière pour ne pas m'en inspirer et créer quelque chose d'original. Une autre raison pour laquelle je n'ai pas pris connaissance de cette application était qu'au départ je pensais qu'il s'agissait d'une fonctionnalité d'un logiciel du laboratoire et qu'il n'était pas possible d'en voir le code. J'avais déjà tracé ma propre voie à ce moment là.

Le second objectif, en parallèle avec le premier, consiste en l'**homogénéisation de la luminosité** sur une méga-image. En effet, si l'on se réfère aux explications de coregistration données précédemment, on apprend qu'une méga-image est composée de plusieurs images composantes. Une image composante correspond à une capture faite avec le microscope, il est donc possible que chaque image composante possède une répartition de la luminosité différente des autres images composantes (due au déplacement de la platine ou encore à la consistance du frotti). L'exemple donné à la figure 3.4 nous montre qu'il est possible de distinguer la séparation entre les images composantes là où ces dernières sont jointes. Ceci est dû à ce problème de luminosité différente sur les diverses captures. Tenter d'homogénéiser cette luminosité améliorera la qualité des méga-images numérisant des frottis sanguins. Là où le premier objectif me faisait prendre connaissance de la partie de visualisation du système de microscopie virtuelle, ce second objectif m'a instruit quant au fonctionnement de la partie d'acquisition et de création des images numériques. En effet, l'homogénéisation est appliquée au moment de la coregistration. Là encore il a fallu plonger dans le code source afin d'en saisir tous les méandres. Ici aussi de nombreuses recherches ont dû être effectuées pour trouver une idée permettant d'homogénéiser les images. Ces recherches sont passées par diverses idées qui n'ont pas toutes été aussi fructueuses.

Une fois une galerie d'images des globules blancs obtenue, il est intéressant de pouvoir proposer une aide au diagnostic. Pour ce faire, il serait idéal de pouvoir remplir l'outil de comptage proposé par C. Jadoul (voir figure 7) sur base des globules blancs détectés dans le frotti. Ainsi, les différents globules seraient comptabilisés dans cet outil. Il faut donc pouvoir classifier les divers globules blancs. En effet, alors qu'il n'y a qu'une seule sorte de globules rouges, les globules blancs sont de plusieurs sortes différentes. Il y a cinq sortes de globules blancs : les lymphocytes, les basophiles, les eosinophiles, les neutrophiles et les monocytes. Le troisième objectif trouve sa traduction dans la **classification des globules blancs** contenus dans une méga-image. Ceci sera utile pour ce qui est appelé la formule leucocytaire qui étudie la proportion des différents globules blancs. Le laboratoire utilise déjà le logiciel CellaVision (voir figure 9) qui permet l'analyse automatique d'images pour la pré-classification des divers globules blancs. Mais ce dernier ne s'applique pas à d'assez grands champs[8].

Cet objectif de classification rencontre un autre objectif qui a été déterminé en cours de stage. En effet, un logiciel utilisé au laboratoire fournissait une base de données qui conte-

[8]On entend par grand champ la numérisation d'une zone conséquente d'un frotti.

nait les images d'une galerie d'images. Cette galerie était aussi une galerie de globules blancs, cependant il était impossible de visionner les images contenues dans la base de données sans utiliser le logiciel l'ayant générée. Il a donc été demandé de pouvoir extraire les images de cette base de données, ce qui a été fait. A partir de là il était intéressant de pouvoir afficher la galerie des images extraites ainsi que les informations importantes reliées à ces images. On rejoint donc la galerie d'images du premier objectif. De plus, à partir des informations de ces images, il était possible de classifier ces images, ce qui rejoint le troisième objectif. Cependant, il fallait pouvoir faire la correspondance entre les images classifiées et les diverses catégories de l'outil de comptage présenté précédemment. L'outil d'extraction des images développé devait donc pouvoir être amélioré pour afficher les images et les classifier tout en remplissant l'outil de comptage. Une autre amélioration à apporter était de pouvoir rendre les images accessibles via DICOM (voir point 2.6).

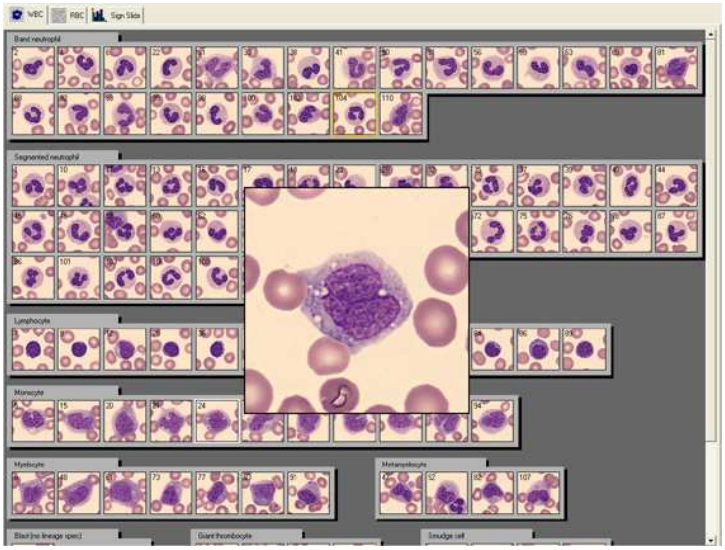

FIG. 9 – Classification des globules blancs avec CellaVision

Un autre objectif au départ du stage était de pouvoir **régler un problème dû à l'autofocus du microscope**. En effet, celui-ci posait problème lorsqu'il s'agissait de faire un focus sur un zone de mise au point uniforme et sans contraste. Il n'arrivait donc pas à obtenir un focus idéal, ce qui rendait les images capturées floues. Ce problème a déjà été souligné dans [Zuy03] et a aussi été abordé par B. Permentier, mémorant à l'IESN, dans [Per06]. Ici encore il a fallu commencer par comprendre le problème et réfléchir à diverses solutions. Un objectif était donc de pouvoir comprendre l'interaction entre AnalySIS et le microscope réel à l'aide du langage imaging-C. Il fallait aussi savoir se servir de ce langage afin d'écrire un module solutionnant ce problème. La réalisation de cet objectif a pris du retard car il y a eu un problème matériel au microscope dont il a fallu attendre le dépannage. Cet objectif a ensuite été abandonné pour des raisons qui seront expliquées au point 3.7.

Voici donc pour les objectifs. En récapitulant, les divers buts à atteindre ont été la création d'une galerie d'images, l'homogénéisation d'image, l'extraction d'images à partir d'une base de données et leur classification, ainsi que solutionner le problème de l'autofocus. Mis à part ce dernier objectif, pour chacun des autres, il a fallu partir de zéro en faisant des recherches et en imaginant des solutions aux problèmes.

p

2 Deuxième partie : matériel et méthode

2.1 Introduction

Le sujet au cœur de cette partie est l'image. En effet, le travail abordé par ce mémoire concerne principalement le traitement d'image. C'est pourquoi nous verrons diverses techniques et méthodes nécessaires à la réalisation de ce travail.

Une image est une représentation bidimentionnelle d'objets tridimensionnels de diverses natures (scène, vue aérienne, paysage, image médicale, etc...). Elle représente l'intensité lumineuse perçue par une caméra (ou tout autre appareil de capture d'image) en chaque point.

Les images numériques sont de plus en plus répandues, notamment grâce à l'apparition sur le marché pour le grand public, depuis quelques années, des scanners et des appareils photos numériques. En informatique, le traitement d'image se pratique sur de telles images et désigne les manipulations automatiques de ces dernières. Le but de ces manipulations est d'obtenir de nouvelles images ou d'en retirer de l'information et cela afin de faciliter l'analyse et l'interprétation d'images, ou encore simplement d'améliorer des images. Un exemple d'amélioration d'image souvent utilisé est la modification de la luminosité d'une photo sous-exposée et donc trop sombre.

Au-delà des manipulations automatisées, le traitement d'image désigne aussi de façon plus globale tous types de traitements appliqués aux images après leur numérisation. Donc, par exemple, la modification d'image, non automatique, à l'aide d'un logiciel spécialisé dans la retouche d'image, en fait partie.

La suite de cette partie présente quelques notions de base de l'imagerie numérique, suivi de quelques descriptions de traitements d'image.

2.2 Notions de base

- **le pixel** :
  Une image numérique est composée d'une multitude de points contigus. Ces points sont des éléments de l'image, en anglais " picture elements " ce qui donne en abrégé : pixel. Ainsi, une image numérique est assimilée à un tableau à deux dimensions donc chaque cellule est remplie par un pixel. Chaque pixel composant l'image est ainsi identifié par des coordonnées X et Y dont la coordonnée [0,0] se situe en haut à gauche de l'image. On peut donc comparer une image numérique à une sorte de mosaïque informatique. Plus le nombre de pixels pour une image est grand, plus cette image sera précise et détaillée et donc plus le tableau la représentant sera grand, ce qui signifie une plus grande nécessité en espace disque.

– **le codage RGB**[9] :
Le codage RGB (Red, Green, Blue) permet de coder la couleur d'un pixel. Ce codage est constitué de trois éléments qui servent de composantes de base pour les couleurs : l'élément R qui désigne le rouge, l'élément G qui désigne le vert et l'élément B qui désigne le bleu. Chacun de ces trois éléments prend une valeur allant de 0 à 255, ce qui signifie qu'il possède 256 intensités différentes. Et donc, chaque combinaison de ces trois éléments donne une couleur comme résultat. Par exemple, pour avoir un rouge vif, il suffit de donner l'intensité maximale à l'élément rouge, donc lui donner la valeur 255, et de ne donner aucune intensité au vert et au bleu, donc leur donner une valeur de 0. Pour obtenir un violet, on combinera donc du rouge et du bleu, ce qui signifiera que la valeur du vert sera faible. Au final, un tel codage fournit 16 793 993 216 combinaisons possibles et donc autant de couleurs possibles.

On retrouve souvent ce codage sous forme hexadecimale, comme par exemple en HTML pour définir la couleur de la police de caractère. En effet, les valeurs de 0 à 255 vont, en hexadecimal, de 00 à FF. Ainsi, un violet codé avec une valeur de 150 pour le rouge, de 50 pour le vert et de 200 pour le bleu nous donne ceci en hexadecimal : 96, 32 et C8. Et donc le codage RGB peut donc être représenté à l'aide de 3 chiffres (d'un octet chacun) ou alors de 6 caractères hexadécimaux (utilisés en HTML). Il est à faire remarquer que ce codage peut parfois posséder un quatrième élément qui représente le canal alpha. Ce dernier élément représente la transparence du pixel prenant lui aussi sa valeur entre 0 et 255. On parle alors de codage RGBA. Ce codage se fait donc sur 4 octets, ce qui, par exemple, en Java se représente par le type int.

Il est à noter que ce système de composition des couleurs est différent du système plus connu par les artistes, qui est un système basé sur la combinaison du cyan, du magenta et du jaune. Ce qui caractérise ces deux systèmes est le caractère tangible ou non. En effet, par exemple, la peinture et l'encre d'une imprimante se basent toutes deux sur le système cyan, magenta et jaune, et sont tangibles, tandis que les couleurs produites par un écran ou un rayon lumineux n'ont rien de tangibles et se basent sur le système rouge, vert et bleu. Comme on peut le voir ci-dessous, la combinaison du rouge, du vert et du bleu à leur intensité maximum, donc avec la valeur 255, produit du blanc, tandis que le noir est composé de ces trois éléments avec la valeur 0.

[9] Voici deux sites Internet très pratiques pour calculer une couleur et d'autres en harmonie avec cette dernière : `http://www.easyrgb.com` et `http://colorblender.com`. A noter que pour l'harmonie des couleurs, la théorie des sept contrastes d'Itten est très intéressante

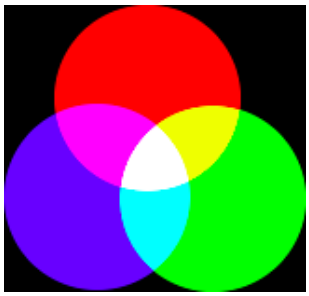

FIG. 10 – Système de codage RGB

– **Niveaux de gris** :

Comme il a été décrit précédemment, un des codages utilisé pour coder une image couleur est le codage RGB. A partir de ce codage, il est possible de transformer une image couleur en image "noir et blanc". Le but est d'obtenir une image dont les pixels prennent leur valeur parmi les niveaux de gris s'étalant du noir au blanc. Le niveau de gris représente l'intensité lumineuse d'un pixel, ainsi une couleur foncée dégagera peu de luminosité et se traduira, en niveaux de gris, par un pixel tirant plus vers le noir. Inversement, une couleur claire sera plus lumineuse et se traduira par un pixel allant plus vers le blanc.

Un pixel en niveaux de gris se traduit par une même valeur dans les trois canaux du codage RGB. Nous avons vu précédemment que le blanc est codé en plaçant chaque composante RGB à la valeur 255 et le noir se voit attribué la valeur 0 à chacune des composantes RGB. De même, tout codage RGB composé de trois valeurs identiques, prises entre 0 et 255, donnera un pixel gris. Les niveaux de gris se répartissent donc sur l'éventail de valeurs allant de 0 à 255.

Pour transformer le codage RGB d'une image couleur en valeur de niveau de gris, plusieurs formules s'offrent à nous. La première et la plus simple est le calcul de la moyenne des trois composantes RGB. Ainsi, la formule utilisée dans ce travail est la suivante : le niveau de gris du pixel $= (R + G + B)/3$. D'autres formules de conversion sont proposées par la Commission Internationale de l'Éclairage (CIE)[10] qui permettent de caractériser la luminance. Une de ces formules est la suivante : le niveau de gris du pixel $= 0.299 \times R + 0.587 \times G + 0.114 \times B$. La somme des trois coefficients vaut 1 et chacun d'entre eux représente la façon dont les couleurs sont perçues par l'œil humain. L'œil humain ne perçoit pas les couleurs avec une égale acuité. En effet, il perçoit préférentiellement le vert, puis le rouge et enfin le bleu, qui est l'ordre retrouvé dans les coefficient de la formule de la CIE. Il suffira ensuite de placer chaque composante RGB à la valeur de gris obtenue, ainsi chacun d'eux aura la même valeur, ce qui donnera le niveau de gris souhaité. Par exemple, pour un rose

[10]Site internet de la CIE : `http://www.cie.co.at`

avec les composantes RGB placées à R=230, G=35 et B=125, on aura une valeur de niveau de gris de 130. Le codage RGB donnant un pixel de ce niveau de gris sera R=130, G=130 et B=130. Le principal avantage d'une telle transformation est que, pour les traitements d'image, il ne faut plus travailler que sur une seule valeur et non sur trois valeurs différentes.

– **L'histogramme** :
Dans le contexte de l'imagerie, le diagramme de fréquence des pixels est appelé histogramme, c'est ce terme qui sera utilisé par la suite pour désigner le diagramme de fréquence.

Un histogramme est un graphique qui représente la fréquence de catégories. Ce graphique possède deux dimensions : l'abscisse représente les diverses classes ou catégories et l'ordonnée représente la fréquence d'apparition de cette catégorie. En traitement d'image, un histogramme sert à représenter le nombre de fois qu'un même pixel apparaît dans une image numérique. C'est donc la fréquence d'un pixel dans l'image. Généralement, l'histogramme d'une image numérique est un histogramme de cette même image en niveaux de gris. Ceci signifie que chaque pixel de l'image a été changé en une valeur de gris allant de 0 à 255. Ainsi, chaque valeur de gris représente une classe sur l'axe X de l'histogramme (donc 256 classes) et l'axe Y représente le nombre de fois qu'un pixel, possédant une certaine valeur de gris, a été rencontré dans l'image. Etant donné que le niveau 0 correspond au noir et le niveau 256 au blanc, il apparaît donc que l'abscise s'étale du plus foncé (à gauche) au plus clair (à droite). Pour une image en couleur, trois histogrammes peuvent être utilisés, un pour chaque composante RGB.

L'histogramme d'une image numérique est souvent utilisé pour appliquer des traitements à cette image comme par exemple l'étirement d'un histogramme, son égalisation, l'obtention du négatif, ou encore le seuillage comme il sera vu plus tard (voir 2.4).

Illustration : à la figure 11, on peut voir une image d'un CD avec, à coté, les histogrammes des trois composantes RGB, et en dessous, l'histogramme en niveaux de gris de cette image.

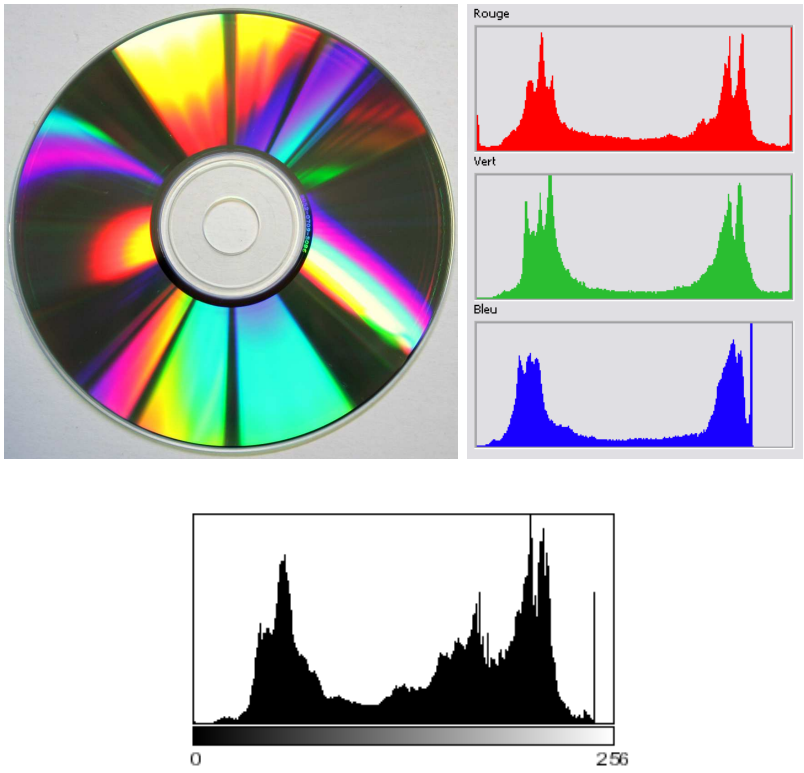

FIG. 11 – Exemple d'histogramme

2.3 Filtrage d'image[11]

Il arrive souvent que les images acquises ne soient pas parfaites, certaines sont floues, d'autres sont perturbées par du **bruit**[12], d'autres encore sont sous-exposées ou surexposées, etc... Le filtrage a pour objectif d'éliminer au mieux les perturbations rencontrées dans une image tout en préservant l'intégrité de la scène d'origine appelée **signal**. Il tend donc à rendre meilleure l'exploitation des images en améliorant le rendu de ces dernières et en corrigeant certains de leurs paramètres (contraste, saturation, ...). Bien que parfois le résultat de certains filtres peut paraître décevant, ceux-ci permettront probablement à l'interprète de discerner certains éléments parmi d'autres. Les filtres consistent à effectuer une opération sur des pixels du voisinage du pixel examiné, y compris lui-même. Ils sont souvent classés en deux catégories : les filtres linéaires et les filtres non-linéaires. Pour filtrer

[11] L'apprentissage de certains traitements d'images s'est fait à l'aide de [Sta04]

[12] On nomme bruit toute perturbation qui affecte la qualité de l'image. Ces perturbations peuvent provenir du dispositif d'acquisition (mauvais réglages du dispositif) ou de la scène elle-même (présence de poussière ou de fumée entre la scène originale et l'objectif, fort éclairage, etc...). Un bon exemple d'image bruitée est la "neige" sur l'image d'un poste de télévision équipé uniquement d'une antenne.

l'image entière, il faut appliquer un algorithme[13] comme suit :

```
Balayage vidéo de l'image
  Si le pixel courant vérifie une condition (optionnelle)
    Balayage vidéo du voisinage centré sur le pixel courant

      Opération sur le pixel

    Fin du balayage local
  Fin si
Fin du balayage de l'image
```

2.3.1 *Filtres linéaires*

Les **filtres linéaires** permettent de supprimer ou de mettre en évidence certaines propriétés de l'image. De tels filtres modifient un pixel en fonction de ses pixels voisins, à l'aide d'une matrice de convolution. Une telle matrice est une matrice carrée (souvent une matrice 3×3 ou une matrice 5×5) qui définit le nombre de pixels voisins permettant la modification du pixel traité, et qui définit la modification à appliquer au pixel traité. Sans entrer dans les détails mathématiques, dans le cas du traitement d'image, la convolution est le traitement d'une matrice par une autre appelée matrice de convolution (ou aussi nommée noyau). La première matrice correspond à une zone de l'image de départ et la matrice de convolution a été définie selon les besoins.

Illustrons le fonctionnement par un exemple simple avec la figure ci-dessous : à gauche se trouve la matrice de l'image, où chaque élément représente un pixel en niveau de gris. A droite se trouve le noyau défini. Finalement, en dessous se trouve le résultat de la convolution. Comment cela fonctionne-t-il ? Le pixel sur lequel est centré la matrice de l'image sera remplacé, dans l'image de résultat, par le produit des deux matrices. Appliqué à chaque pixel de l'image initiale, une tel noyau a pour effet de la décaler d'un pixel vers la droite. Dans la figure 12, pour obtenir le résultat du pixel central (en rouge), il faut multiplier chaque pixel du noyau par le pixel correspondant de l'image de départ. En prenant la somme de chacune de ces multiplications, on obtient la valeur que contiendra le pixel de l'image résultat. Il vient donc que $(0 \times 1) + (0 \times 2) + (0 \times 3) + (1 \times 4) + (0 \times 5) + (0 \times 6) + (0 \times 7) + (0 \times 8) + (0 \times 9) = 4$. Remarque : les résultats sont stockés dans l'image résultat qui est une copie, le calcul de la convolution se fait sur base des valeurs d'origine pour chaque pixel et non sur les valeurs modifiées par le filtre. En formule mathématique, voici ce que ça donne :

$$F(x,y) = \sum_{i=0}^{N} Noyau(i,j) \ \times \ Image(x+i-1, y+j-1)$$

[13] Dans l'algorithme, "balayage vidéo" signifie parcourir l'image dans son entièreté de gauche à droite et de bas en haut

Où N est l'ordre du noyau moins 1. Dans notre exemple ci-dessous, le noyau est une matrice 3×3, N sera donc de $3 - 1 = 2$

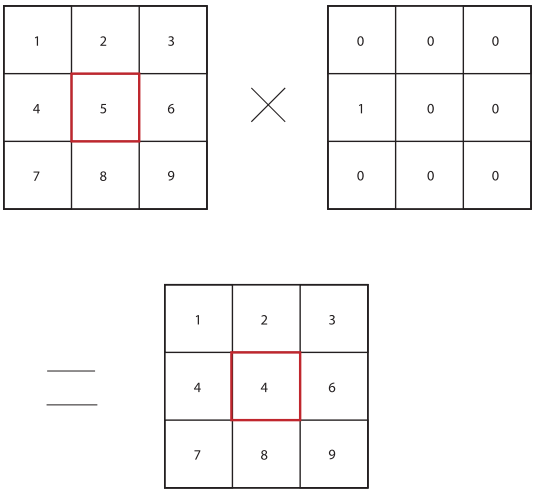

FIG. 12 – Exemple de filtre de convolution

Les filtres linéaires se divisent en deux catégories :
- **Le filtrage passe-bas** qui consiste en l'atténuation des variations de luminance[14]. L'effet obtenu est un lissage de l'image via la limitation des brusques variations de luminance. Ceci a pour effet d'atténuer le bruit.
- **Le filtrage passe-haut**, au contraire, permet de rehausser les variations de luminance ce qui améliore le contraste. Ceci a pour effet de renforcer les contours.

2.3.2 *Filtres non-linéaires*

La catégorie **filtres non-linéaires** qui comprend les filtres morphologiques (étant donné que ces filtres ne sont pas utilisés dans ce travail, ceux-ci seront brièvement décrits et illustrés). Le filtre médian est aussi un filtre qui entre dans cette catégorie. Le filtre Sobel, le filtre Prewitt et le filtre Laplacien (voir 2.3.4) sont aussi considérés comme étant non-linéaires.

Les **filtres morphologiques** sont des filtres basés sur la morphologie mathématique qui utilise la théorie des ensembles. Le principe est de comparer l'image originale à un ensemble connu appelé *élément structurant*. Cet élément structurant peut être vu comme une forme de référence composée d'un ensemble de pixels. Il possède une taille connue et un centre connu. A la base, ces filtres traitent des images binaires (voir 2.4.4) mais sont aussi applicables aux images en niveaux de gris. L'élément structurant est déplacé dans toute l'image pour que son centre passe par chaque pixel de celle-ci. A chaque position du centre de l'élément structurant, il faut confirmer ou infirmer sa présence dans la forme de

[14] peut-être faut-il définir luminance

référence à l'aide d'opérateurs ensemblistes tels que l'union ou l'intersection.

Les deux opérations morphologiques de base sont l'érosion et la dilatation.

- **L'érosion** va éliminer tout ce qui est de la taille inférieure à celle de l'élément structurant. Les formes de taille inférieure disparaîtront, les formes de taille supérieure verront leur taille diminuer et les éventuels trous dans les formes seront accentués. Certaines liaisons entre formes pourront disparaître.
- **La dilatation**, à l'inverse, augmente la taille des formes en fonction de la taille de l'élément structurant. Les éventuels trous dans les formes se rempliront et des formes proches l'une de l'autre pourront être reliées.

A partir de ces deux opérateurs, deux autres opérateurs peuvent être obtenus. Il s'agit de l'ouverture et de la fermeture.

- **L'ouverture** est composée d'une érosion suivie d'une dilatation. Cet opérateur permet d'éliminer les formes de taille inférieure à celle de l'élément structurant.
- **La fermeture** est composée d'une dilatation suivie d'une érosion. Cet opérateur permet de combler les éventuels trous de taille inférieure à celle de l'élément structurant.

Un autre opérateur des filtres morphologiques est la squelettisation qui permet d'obtenir le squelette des formes (attention, il ne s'agit pas des contours des formes). Un exemple d'utilité de la squelettisation est la reconnaissance d'écriture.

Le **filtre médian** permet le débruitage. Le fonctionnement d'un tel filtre consiste à classer les valeurs des pixels du voisinage, le pixel étudié y compris, par ordre croissant. Ensuite il affecte au pixel étudié la valeur médiane du classement obtenu. Un tel filtre possède l'avantage de ne pas modifier la luminance car la valeur de substitution n'est pas une moyenne mais une valeur d'un voisinage. De plus les frontières entre les objets ne sont pas décalées par un effet de flou. L'algorithme codé en Java du filtre médian est illustré à l'annexe G.

2.3.3 *Quelques filtres*

A la figure 13 se trouve une illustration de quelques filtres. La première image du tigre est l'image originale (1). En dessous de cette dernière, de gauche à droite se trouve cette même image filtrée avec les filtres suivants : filtre moyen (filtre linéaire, passe-bas), filtre gaussien (filtre linéaire, passe-bas), filtre kuwahara (filtre non-linéaire) et un filtre médian (filtre non-linéaire).

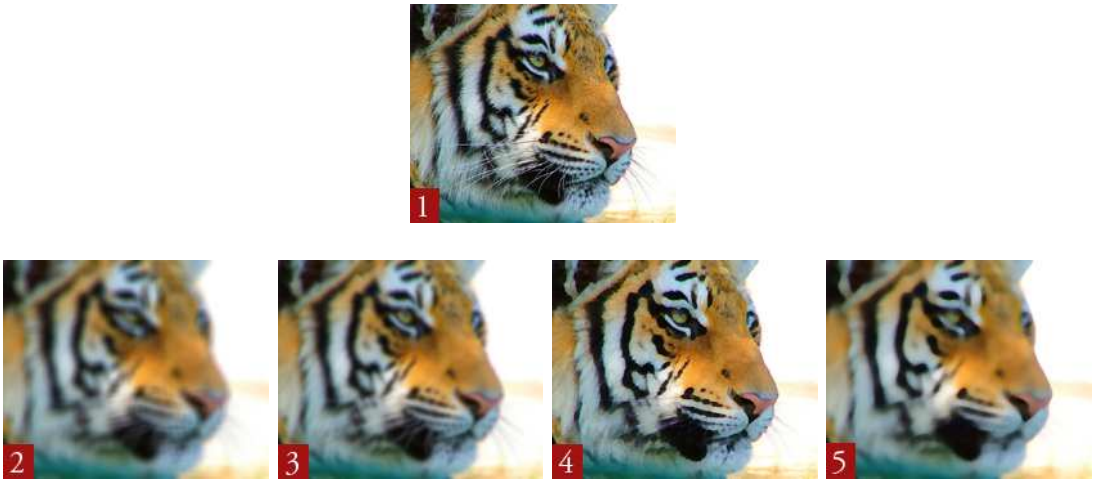

FIG. 13 – Exemple de filtres : moyen (2), gaussien (3), kuwahara (4) et médian (5)

Voici un exemple d'utilité du filtre médian : la première image est l'image du CD déjà rencontrée à la figure 11 à laquelle du bruit a été ajouté (des points blancs et des points noirs ont été disséminés de façon aléatoire). Juste à côté, l'effet du filtre médian appliqué à cette image peut être observé.

FIG. 14 – Exemple de filtre : utilité d'un filtre médian

Illustrons maintenant les filtres morphologiques. Comme il a été vu, l'ouverture permet d'éliminer des formes et est composée d'une érosion suivie d'une dilatation. Dans l'exemple de la figure 15, on peut observer deux dés dans l'image d'origine. Une fois l'érosion appliquée, le dé de droite a disparu. Enfin, la dilatation reconstruit le dé de gauche et le dé de droite n'est pas réapparu.

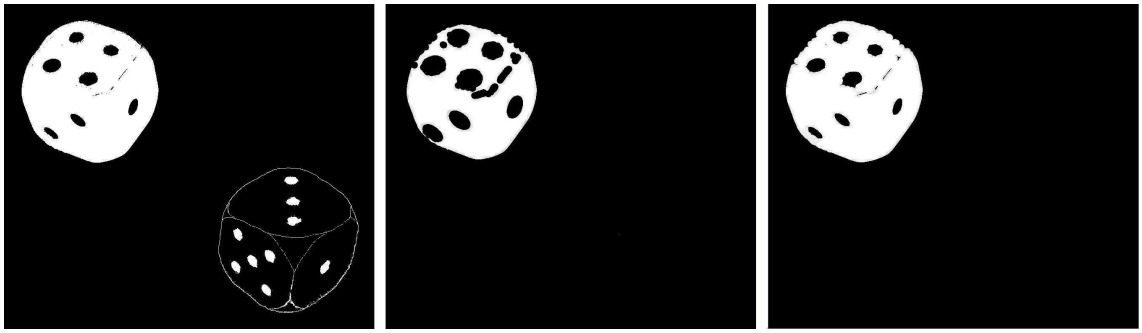

FIG. 15 – Exemple de filtre morphologique : ouverture

2.3.4 *Détection de contours et chaînage de pixels*

Visuellement, les contours délimitant une forme nous paraissent comme un indice visuel intuitif et idéal pour l'analyse et l'interprétation des images. La détection de contours consiste à détecter les bords des formes contenues dans une image numérique. Ces formes sont supposées être séparées par des contours continus qui permettent de les identifier les unes des autres. Une fois la détection de contours appliquée à une image numérique, le résultat obtenu fournit une représentation schématique de l'image. Cette image est une abstraction et une simplification de l'image d'origine mais qui préserve la structure de cette dernière.

De manière automatisée, pour la **détection de contours**, les contours se caractérisent par un variation importante de la luminosité. Par exemple, si on observe une image représentant une maison de face, supposons le mur en crépi et le toit en tuiles foncées, alors le passage de la zone claire du crépi à la zone foncée des tuiles se traduira par une discontinuité de l'intensité lumineuse et pourra être interprétée comme étant le contour délimitant le toit. Généralement, cette détection de variation est appliquée aux images en niveaux de gris, ce qui signifie dans ce cas qu'un contour est identifié par une variation significative du niveau de gris. Concrètement, pour mesurer cette variation, deux techniques existent : le calcul du gradient et le calcul de la dérivée seconde.

A l'aide du **calcul du gradient**, la détection de contours se fait en calculant la norme du gradient en chaque point de l'image. Lorsque la valeur de cette norme est maximale, on obtient alors un contour. Cette technique fonctionne de façon optimale lorsque la transition entre les niveaux de gris est abrupte. Les filtres de convolution Sobel, Prewitt, Kirsh ou encore Canny-Deriche sont des exemples de filtres utilisant cette technique.

A l'aide de **la dérivée seconde**, le contour est caractérisé par un passage par zéro. cette technique est avantageuse quand la transition se fait sur une région plus large. L'exemple type d'utilisation de cette technique est le filtre de convolution Laplacien qui est beaucoup plus sensible que les précédents. Le Laplacien est une dérivée seconde à deux dimensions,

ce qui donne :

$$L(x,y) = \frac{\partial^2 Image(x,y)}{\partial x^2} + \frac{\partial^2 Image(x,y)}{\partial y^2}$$

Il est à noter que la somme des éléments d'un noyau Laplacien est toujours nulle. Voici les 3 noyaux différents que l'on peut rencontrer pour ce filtre.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \text{ et } \begin{bmatrix} 1 & -2 & -1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Voici quelques exemples de filtres de détection de contours appliqués à cette image d'une guitare : de gauche à droite nous avons les filtres Sobel sur la première ligne, Prewitt et Kirsh sur la seconde et Canny-Deriche et laplacien sur la dernière.

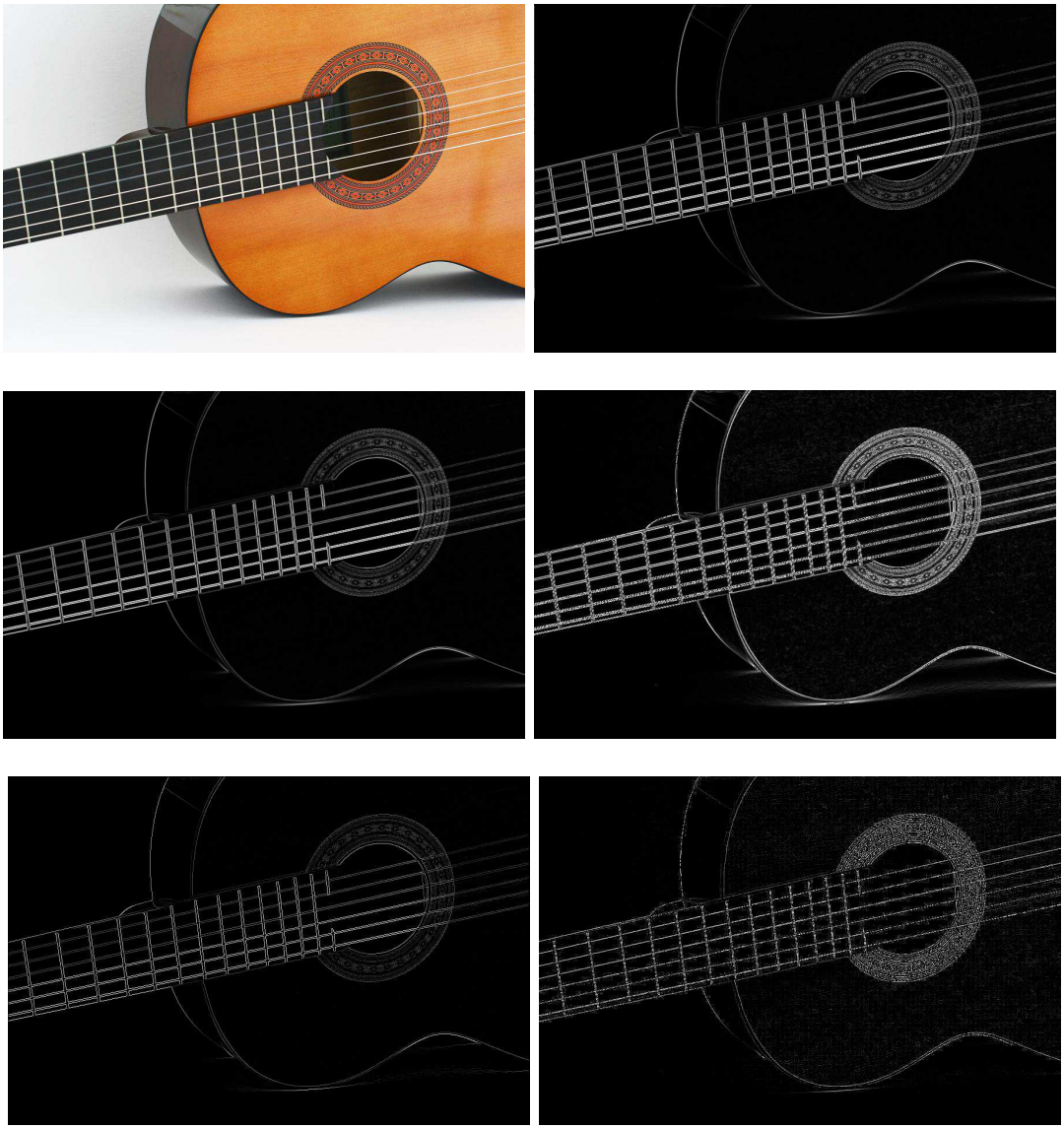

FIG. 16 – Exemple de filtres : détection de contours (Sobel, Prewitt, Kirsh, Canny-Deriche et Laplace)

Bien entendu, pour la détection de contours, on suppose que l'image est composée d'éléments parfaitement homogènes, ce qui n'est pas souvent le cas. Le bruit posera souvent problème car il fera partie des contours. Prenons par exemple un objet dont une partie est à l'ombre et l'autre au soleil, sur l'image ceci se traduira par une frontière entre la zone ombrée et la zone illuminée, cette frontière sera prise en compte dans les contours détectés, or il s'agit d'un seul et même objet. Résultat : cet objet sera coupé en deux par un contour qui délimitera l'ombre de la lumière. De même, il est possible que la frontière entre un objet et l'arrière plan devienne imperceptible, d'où la possible disparition d'une partie du contour détecté. L'information obtenue par la détection de contours est donc souvent lacunaire et bruitée. Généralement il faut passer par des pré-traitements sur l'image de façon à l'épurer au mieux afin d'obtenir un résultat idéal pour la détection de contours.

Une fois les pixels appartenant aux contours mis en évidence, l'image obtenue est une image binaire (image contenant uniquement deux couleurs, voir 2.4.4) dont, idéalement, les contours sont affinés pour atteindre une épaisseur d'un seul pixel. L'image binaire nous permet de savoir si un pixel appartient au contour ou non. A partir de cette image, on désire obtenir des structures et des formes. En effet, avec une telle image on ne peut pas dire à quel contour appartient un pixel ou encore quel est le pixel suivant ou précédant celui-ci dans le contour. L'étape suivante qui nous permettra de connaître ces informations est le **chaînage de pixels**. Le chaînage permet d'obtenir une liste de contours. Chaque contour de la liste appartient à une forme. Par exemple, une image contenant un carré et un cercle aboutira à une liste de 2 éléments. Le premier sera le contour du carré et le second celui du cercle. Chaque contour est une liste de pixels. Le chaînage de pixels nous permet donc d'avoir une liste de listes de pixels et nous permet d'avoir une information contextuelle entre les pixels alors que l'image binaire n'était, au final, qu'une matrice. Lors du chaînage, un contour est appelé une chaîne. Une chaîne possède un début, une fin et une longueur. Il est possible d'éliminer une chaîne si cette dernière n'est pas suffisamment longue, ce qui permet d'éliminer du bruit. Il est aussi possible de fusionner plusieurs chaînes. Il existe plusieurs techniques de chaînage des pixels, nous citerons juste l'approximation polygonale, qui consiste à remplacer un contour par une ligne polygonale, et nous décrirons le chaînage par automate ainsi que le chaînage parallèle.

Le chaînage par automate consiste à extraire une chaîne dès qu'un contour est rencontré. Donc, dès qu'un pixel appartenant à un contour est rencontré, on extrait le contour en entier. Voici l'algorithme :

```
Balayage vidéo de l'image binaire
  Si le pixel courant appartient à un contour

    extraction complète du contour point par point
    reprise du balayage là où il s'était arrêté

  Fin si
Fin du balayage de l'image
```

Il faut faire attention à marquer les pixels des contours déjà extraits de manière à ne plus les prendre en compte dans la suite du balayage. Cet algorithme pose quelques problèmes : si le pixel rencontré n'est pas une extrémité du contour, quelle direction suivre pour l'extraction ? S'il y a un embranchement, quelle branche suivre ?

Le chaînage parallèle part du principe qu'un contour possède deux extrémités, une tête et une queue. L'algorithme impose qu'on ne peut accroître un contour qu'à partir de sa queue. Pour un contour fermé, alors un pixel peut être à la fois tête et queue. Ce type de chaînage fait aussi intervenir les notions de passé et de futur du pixel. Dans le cas d'un balayage de l'image de gauche à droite et de haut en bas, le passé d'un pixel est constitué des pixels adjacents à ce dernier et se situant directement à sa gauche, au-dessus à gauche, au-dessus et au-dessus à droite. Le futur est constitué des pixels adjacents directement à droite, en bas à gauche, en bas et en bas à droite. Le passé et le futur d'un pixel ainsi que lui-même forment une matrice 3×3. Le passé correspond aux pixels déjà examinés par l'algorithme et le futur à ceux qui n'ont pas encore été examinés. L'algorithme de ce chaînage provient de Gérard Giraudon[15] et se construit comme suit :

```
Balayage vidéo de l'image binaire
  Si le pixel courant appartient à un contour
    Si ce même pixel est le 1er du contour
    (donc s'il n'a pas d'extrémité queue dans son voisinage
    passé)

      Créer un nouveau contour
      Y inclure le pixel courant
      (celui-ci devient tête et provisoirement queue)

    Sinon s'il y a une extrémité queue dans le voisinage passé

      Inclure le pixel dans le contour de son voisinage passé
      (il devient la nouvelle queue)

    Sinon s'il y a plusieurs extrémités de contour

      Inclure le pixel dans un des contours
      Fusionner les contours en un seul

    Fin si
  Fin si
Fin du balayage de l'image
```

Le terme parallèle de ce chaînage provient du fait que plusieurs chaînes sont extraites en même temps et non une à une comme dans le chaînage par automate. La matrice qui suit

[15] Pour plus de détails se référer à [Gir87]

illustre le passé (P) et le futur (F) d'un pixel : $\begin{bmatrix} P & P & P \\ P & Pixel & F \\ F & F & F \end{bmatrix}$.

2.4 Segmentation et techniques de seuillage d'image

Il est courant, en traitement d'image, de vouloir extraire les divers objets présents dans une image. C'est le but recherché de la segmentation qui tente de découper l'image en domaines qui correspondent aux objets composants cette image. La segmentation permet ainsi à l'analyse d'image de séparer les objets importants à cette analyse et d'extraire les régions d'intérêt du fond de l'image. On cherche donc à donner une étiquette à chaque pixel de l'image, de manière à pouvoir l'identifier à un objet de l'image. Il existe plusieurs techniques permettant de déterminer des classes dans une image, ce chapitre s'attardera sur la classification par **seuillage**.

L'objectif recherché par le seuillage est la détermination de classes dans une image en se basant sur l'histogramme de cette image. Le seuillage peut être manuel ou automatique, le but étant de déterminer un ou plusieurs seuils qui serviront de valeurs de partition de l'image, il est plus simple de travailler avec une image en niveaux de gris. En effet, une telle image est caractérisée par un seul histogramme sur lequel on pourra travailler. Ainsi, les techniques de seuillage font l'hypothèse que chaque pic de l'histogramme représente une classe d'objet. Il faut donc déterminer les seuils qui séparent les pics.

Voici une explication par l'exemple :
A la figure 17, une image convertie en niveaux de gris (256 niveaux) et son histogramme associé. Cet histogramme se compose de trois pics, le premier à gauche représente les couleurs foncées, donc l'ombre de l'arrière plan, le second au centre et étalé représente les framboises (mais aussi certaines zones ombrées des doigts), et le troisième à droite représente les couleurs claires, c'est-à-dire la main. Deux seuils peuvent donc être placés pour séparer ces trois pics, un est placé au niveau de gris de valeur 35 et un autre au niveau de valeur 160. Une fois les seuils appliqués, l'image qui en résulte est une image composée de trois couleurs, une pour chaque pic. Le rose clair est utilisé pour la main, le rouge correspond aux framboises ainsi que certaines ombres des doigts, et finalement le noir représente l'arrière plan sous la main.

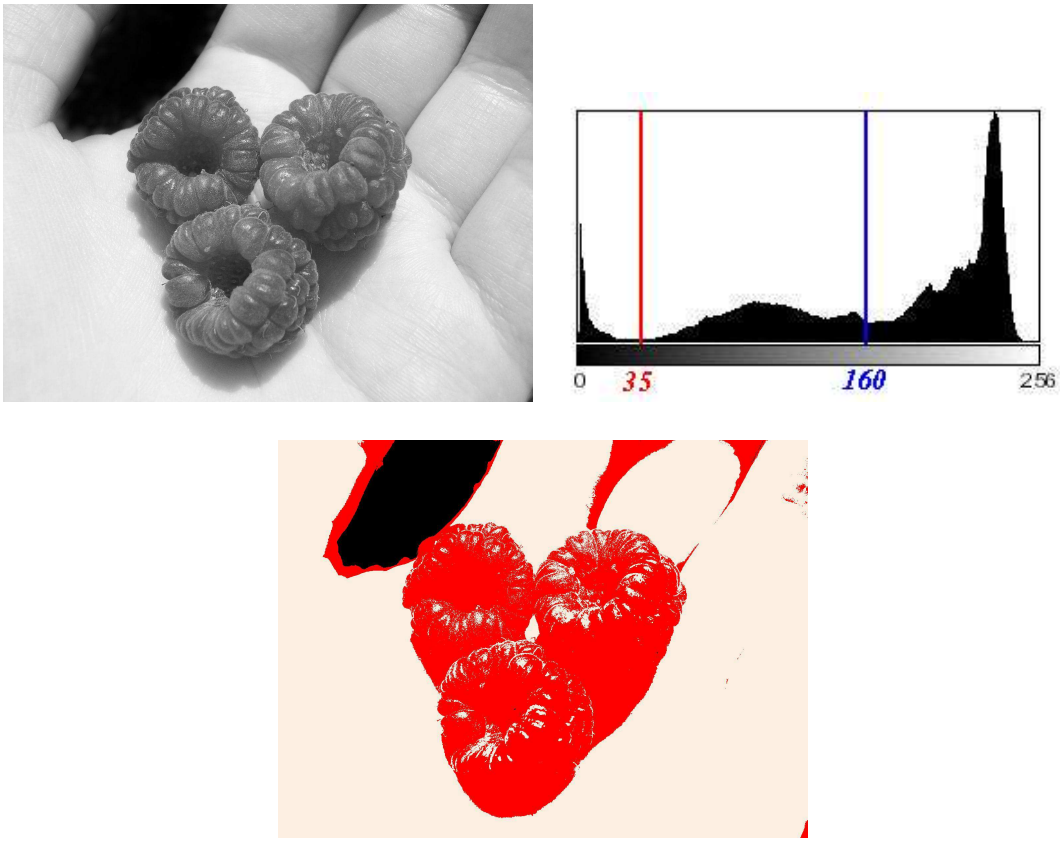

FIG. 17 – Exemple de seuillage

Deux cas de seuillage peuvent se présenter, le seuillage simple et le seuillage multiple :

- **Le seuillage simple** consiste à donner une même couleur à tous les pixels se situant au-dessus d'un seuil. Le résultat d'un tel seuillage donnera une image divisée en deux classes distinctes. Une telle image est dite "binaire" (voir plus loin : "Binarisation d'images")
- **Le seuillage multiple** prend en compte plusieurs seuils faisant office de bornes délimitant les classes. Ainsi, les pixels de l'image se verront attribuer une couleur en fonction des bornes (ou seuils) entre lesquelles ils se situent. L'image résultat sera donc composée d'autant de couleurs qu'il y a de classes.

Le seuillage étant basé sur l'histogramme, comment déterminer les divers seuils ? Une fois l'histogramme construit, il suffit de placer les seuils au plus bas des vallées présentes dans l'histogramme. De façon manuelle, on peut visualiser ces vallées et donc voir où situer les seuils. Cependant, de façon automatique ce n'est pas chose évidente et la technique idéale de seuillage n'existe pas. Beaucoup de ces techniques fonctionnent correctement sur base d'histogrammes dont les pics sont séparés de façon bien distincte. De plus, ces techniques sont loin d'être efficaces dans tous les cas de figure et ne sont donc pas à généraliser à tous les cas multi-classes. Les points qui suivent visent à présenter diverses techniques de détermination automatique de seuils.

2.4.1 *Seuillage d'Otsu*

La méthode d'Otsu[16] est une méthode de calcul de seuil automatique divisant une image en deux classes, typiquement l'arrière-plan et les objets. Cette méthode travaille donc sur un histogramme bimodal, c'est-à-dire un histogramme possédant 2 pics. Selon cette méthode, le meilleur seuillage, permettant de séparer l'histogramme en deux groupes de pixels, est déterminé par un seuil qui correspond à une minimisation de la variance **intra-classes** et une maximisation de la variance **inter-classes**.

Le but étant de séparer les deux classes d'un histogramme bimodal, il faut donc déterminer le seuil séparant ces deux classes. Idéalement, chaque classe (donc chacun des deux pics de l'histogramme) devrait avoir une forme de cloche bien lisse, ce qui signifierait que chacune de ces deux classes est homogène et bien distincte l'une de l'autre.

La variance est nécessaire pour mesurer l'homogénéité (l'homogénéité se traduit par le fait que les éléments d'une classe sont regroupés autour de la moyenne de cette classe) d'un groupe. En effet, un groupe homogène possèdera une faible variance, et inversement, un groupe avec une faible homogénéité possèdera une variance élevée. Ainsi, pour déterminer le seuil optimal via la méthode d'Otsu, il faut minimiser la variance intra-classes. De même, une seconde façon d'obtenir ce seuil optimal sera de maximiser la variance inter-classes de manière à séparer les classes. Chacune de ces deux façons conduit au même résultat car la variance totale (la somme des variances inter et intra) est constante. Ainsi, il suffit de rechercher de façon séquentielle, parmi toutes les valeurs de seuil possibles, celle qui minimise la variance intra-classes. La valeur trouvée sera aussi celle qui maximisera la variance inter-classes. Finalement, Otsu assimile donc le problème de recherche d'un seuil pertinent à un problème de classification des pixels en deux groupes.

Mathématiquement, la méthode d'Otsu s'exprime comme suit : supposons l'histogramme divisé en deux classes. La première classe s'étale du début de l'histogramme jusqu'à t et la seconde de t jusqu'à la fin de l'histogramme, que nous appellerons I. On peut alors définir :

– **La probabilité de classe** :

$q_1(t)$ la probabilité pour la classe inférieure ou égale à t :

$$q_1(t) = \sum_{i=0}^{t} P(i)$$

$q_2(t)$ la probabilité pour la classe supérieure à t :

$$q_2(t) = \sum_{i=t+1}^{I} P(i)$$

[16] Pour plus d'informations sur cette méthode, s'en référer à [Ots79]

– **La moyenne de la classe** :

$\mu_1(t)$ la moyenne pour la classe inférieure ou égale à t :

$$\mu_1(t) = \sum_{i=0}^{t} i \ \frac{P(i)}{q_1(t)}$$

$\mu_2(t)$ la moyenne pour la classe supérieure à t :

$$\mu_2(t) = \sum_{i=t+1}^{I} i \ \frac{P(i)}{q_2(t)}$$

– **La variance de la classe** :

$\sigma_1^2(t)$ la variance pour la classe inférieure ou égale à t :

$$\sigma_1^2(t) = \sum_{i=0}^{t} [i \ - \ \mu_1(t)]^2 \ \frac{P(i)}{q_1(t)}$$

$\sigma_2^2(t)$ la variance pour la classe supérieure à t :

$$\sigma_2^2 = \sum_{i=t+1}^{I} [i \ - \ \mu_2(t)]^2 \ \frac{P(i)}{q_2(t)}$$

– **La variance intra-classes** $\sigma_\omega^2(t)$:

$$\sigma_\omega^2(t) = q_1(t)\sigma_1^2(t) \ + \ q_2(t)\sigma_2^2(t)$$

– **La variance inter-classes** $\sigma_\beta^2(t)$:

$$\sigma_\beta^2(t) = q_1(t)[1 \ - \ q_1(t)][\mu_1(t) \ - \ \mu_2(t)]^2$$

Il reste à parcourir l'ensemble des valeurs de t ([0,255] pour un histogramme d'une image en niveaux de gris) et à en trouver celle qui minimise $\sigma_1^\omega(t)$. Cette algorithme opérant directement sur les niveaux de gris est assez rapide une fois l'histogramme construit. Il est à noter qu'il sera efficace dans le cas d'un histogramme à distribution bimodale des valeurs de gris. Dans le cas contraire, le résultat sera probablement inexploitable.

Illustration :
L'image de la figure 18 a été modifiée de façon à éclaircir les zones très sombres, ainsi le pic de gauche de l'histogramme fais une translation pour s'inclure dans le pic de droite. De ce fait, l'histogramme peut être considéré comme bimodal. L'algorithme d'Otsu de détection automatique du seuil nous donne un seuil à 162. L'image résultat à laquelle le seuil a été appliqué nous montre en rouge tout ce qui se trouve sous ce seuil, c'est-à-dire les framboises et certaines ombres.

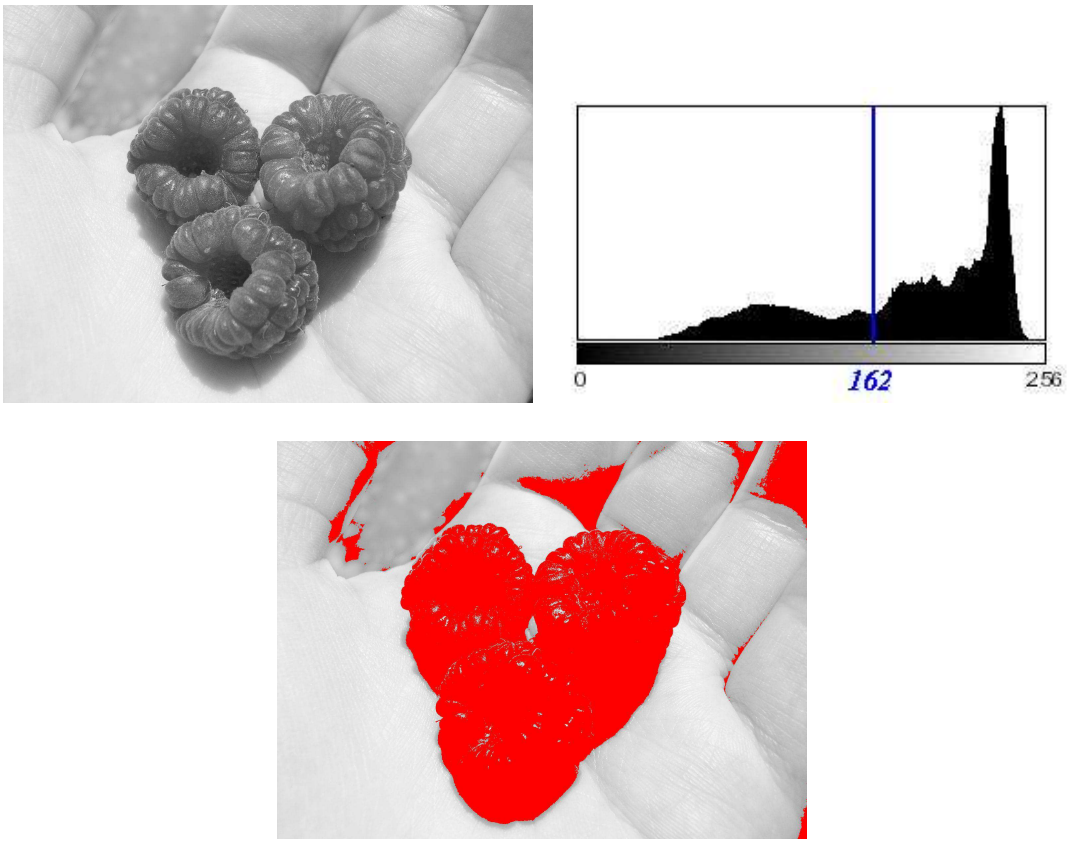

FIG. 18 – Exemple algorithme d'Otsu

L'algorithme codé en java se trouve à l'annexe F.1.

2.4.2 *Seuillage entropique*

Le seuillage entropique nous vient de la théorie de l'information et plus précisément à **l'entropie de Shannon** qui correspond à la mesure de l'incertitude liée à un événement aléatoire. Cette méthode détermine le seuil en maximisant l'entropie sur base de l'histogramme des niveaux de gris. Elle considère que l'arrière-plan et l'avant-plan d'une image proviennent tous deux d'une source différente. Elle considère donc qu'il y a deux classes dans l'histogramme. Sur base d'un tel histogramme, le seuil séparant l'arrière-plan

de l'avant-plan est obtenu lorsque la somme de l'entropie des deux classes est maximisée. Ainsi, pour trouver le seuil optimal, il suffit de rechercher de façon séquentielle, parmi toutes les valeurs de seuil possibles, celle qui maximise la somme de l'entropie des deux classes séparées par cette valeur. Le but ici est d'obtenir deux classes possédant toutes deux un maximum d'informations.

Mathématiquement, la méthode de seuillage entropique s'exprime comme suit : supposons l'histogramme divisé en deux classes. La première classe s'étale du début de l'histogramme jusqu'à t et la seconde de t jusqu'à la fin de l'histogramme, que nous appellerons I. Dans ce cas-ci, il faut que l'histogramme soit normalisé, c'est à dire que $\sum_{i=0}^{I} h_i = 1$. On peut alors définir :

– **L'entropie de classe** :

$H_1(t)$ l'entropie pour la classe inférieure ou égale à t :

$$H_1(t) = -\sum_{i=0}^{t} h_i \ln h_i$$

$qH2(t)$ l'entropie pour la classe supérieure à t :

$$H_2(t) = -\sum_{i=t+1}^{I} h_i \ln h_i$$

Il reste à parcourir l'ensemble des valeurs de t ([0,255] pour un histogramme d'une image en niveaux de gris) et en retirer celle qui maximise $H_1 + H_2$.

Illustration :
Reprenons notre exemple pour la figure 19. L'algorithme de l'entropie maximum pour la détection automatique du seuil nous donne un seuil à 160. L'image résultat à laquelle le seuil a été appliqué nous montre en rouge tout ce qui se trouve sous ce seuil, c'est-à-dire les framboises et certaines ombres.

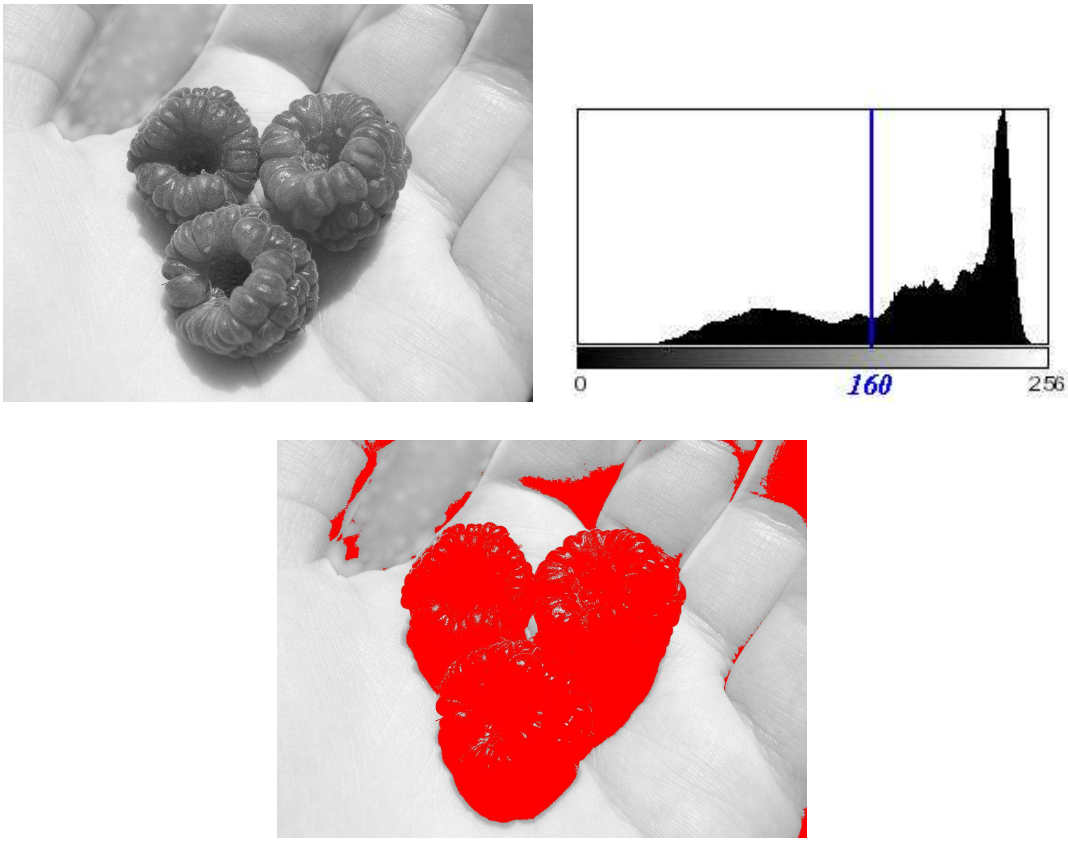

FIG. 19 – Exemple algorithme d'entropie maximum

L'algorithme codé en java se trouve à l'annexe F.2.

2.4.3 *Autres approches*

Bien entendu il existe d'autres techniques de segmentation. La méthode d'Otsu et la méthode de l'entropie maximum sont des méthodes de segmentation qui se basent sur l'histogramme de l'image que l'on peut nommer **approche globale** souvent désigné par classification. Il existe aussi d'autres approches de segmentation dont **l'approche région** (region-based segmentation), qui recherche les composants homogènes, et **l'approche contour**, parfois appelée approche frontière (edge-based segmentation), qui recherche les contours des objets.

- **l'approche région** :
  Cette segmentation vise à découper l'image en régions homogènes. Une région correspond à un regroupement de pixels adjacents ayant des propriétés communes ou plus précisément dont les attributs ne varient pas beaucoup. Ainsi, une région sera caractérisée par un critère à vérifier appelé critère d'homogénéité, par exemple que la variance d'une région soit inférieure à une certaine valeur. A partir de là, la région

va s'agrandir en y agrégeant les pixels adjacents qui n'empêcheront pas la région de remplir la condition donnée. L'algorithme le plus souvent cité mettant en oeuvre cette approche est celui de division et fusion (split and merge) qui date de 1974[17]. Cet algorithme procède d'abord à une division de l'image en régions. Ensuite il fusionne des régions adjacentes de sorte que certaines conditions soient vérifiées, comme par exemple un nombre fixé de régions à atteindre. Un autre algorithme est celui de croissance de région (region growing) dont voici une illustration de résultat :

FIG. 20 – Exemple algorithme *region growing*

– **l'approche contour** :
Cette approche exploite le fait que deux régions connexes sont séparées par un contour qui sert de transition entre ces deux régions. Ces contours se manifestent par une discontinuité, que ce soit dans les niveaux de gris, dans les couleurs ou encore dans les textures. L'avantage principal de cette approche provient des modèles déformables, en effet dans le cas d'une suite d'images variant peu l'une de l'autre, il n'est pas nécessaire de recalculer tous les contours des régions mais de repartir des résultats obtenus dans une image précédente et de déformer les contours pour coller à la nouvelle image.

2.4.4 *Binarisation d'images*

La binarisation consiste à transformer une image en image binaire, c'est-à-dire une image dont les pixels sont typiquement, soit noirs, soit blancs (bien entendu, tout autre couple de couleurs peut-être utilisé pour binariser une image). Pour ce faire, il faut déterminer un seuil qui séparera les pixels blancs des pixels noirs. L'opération de binarisation divisera donc l'image en deux classes distinctes, généralement une classe pour l'arrière-plan et une classe pour les objets. L'avantage de cette opération est de fournir une image résultat qui est très simplifiée et donc, à priori, plus simple à traiter.

[17] Voir [PH74]

Mathématiquement, on peut définir la binarisation comme suit :

$$imageBinaire(x,y) = \begin{cases} noir & \text{si} \quad image(x,y) \leq t \\ blanc & \text{sinon} \end{cases}$$

Où t est le seuil séparant les deux classes.

Illustration :
Toujours avec notre exemple, voici à la figure 21 la binarisation appliquée au seuil obtenu par la méthode d'entropie maximum (160).

FIG. 21 – Exemple de binarisation

2.4.5 *Conclusion sur le seuillage et la segmentation*

Précédemment, nous avons examiné deux techniques de seuillage simple, le seuillage d'Otsu et le seuillage d'entropie maximum. Les exemples illustrant ces deux techniques offrent des résultats quasi similaires. Cependant, il est à noter que le cas illustré se prête bien à la similitude des résultats obtenus, or certains cas ne fournissent pas du tout les mêmes résultats.

Illustration par l'exemple :
A la figure 22 nous avons un lampadaire suivi de son image correspondante en niveaux de gris. Juste en dessous, nous avons, à gauche, l'image dont la partie se trouvant sous le seuil obtenu est colorée en rouge. A gauche le seuil est obtenu via le seuillage d'Otsu (130) et à droite via le seuillage d'entropie maximum (55).

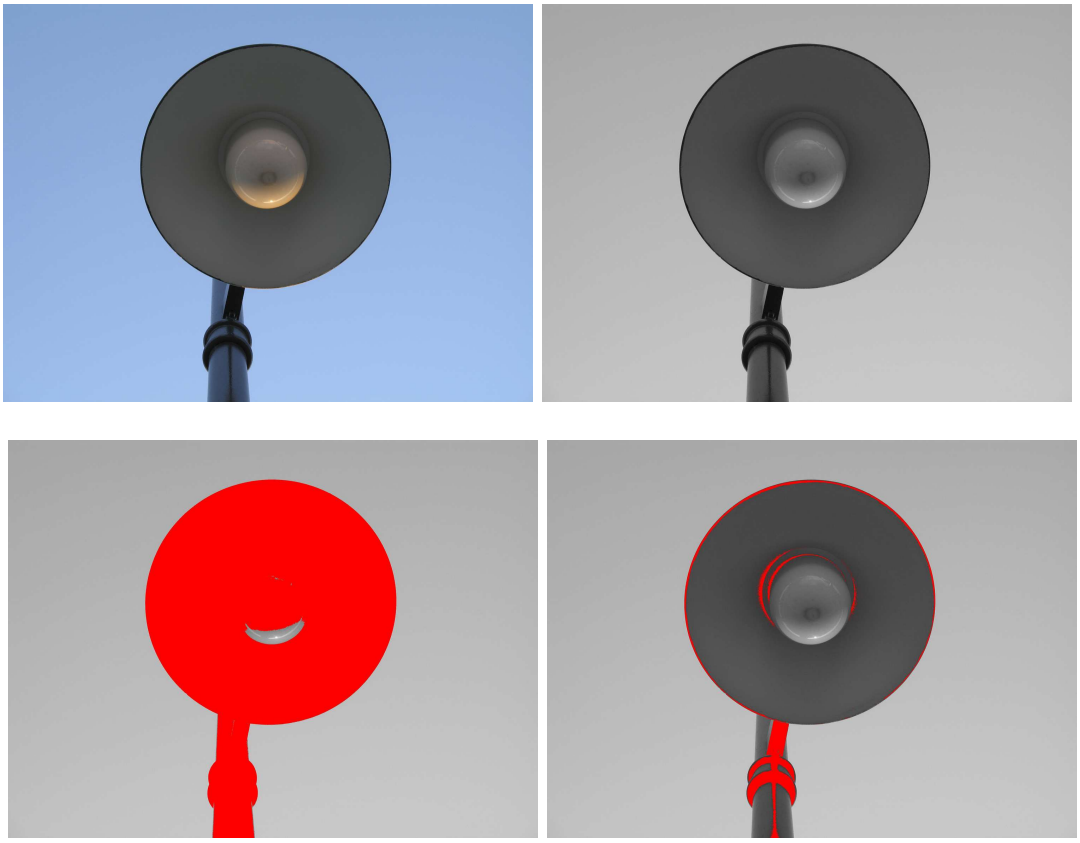

FIG. 22 – Différence entre Otsu (à gauche) et entropie maximum (à droite)

Le seuillage d'une image nécessite de trouver un ou plusieurs seuils. La méthode d'Otsu et la méthode de l'entropie maximum (ainsi que d'autres telles que isoData, superposition de deux gaussiennes, etc) permettent d'obtenir un seuil qui divisera l'image en deux classes. Un tel seuil peut être utile pour la binarisation d'une image. Cependant, pour que ces techniques produisent un résultat pertinent, l'image de départ doit respecter la contrainte d'un contraste suffisamment marqué entre l'arrière-plan et les objets de la scène à extraire. De plus, beaucoup de méthodes ne sont pas généralisables au seuillage d'images à plus de deux classes. Et de surcroît, les méthodes qui sont, à priori, adaptées au seuillage multiple (comme par exemple le seuillage par détection de vallées), ne donneront pas toujours des résultats optimaux étant donné qu'il est très difficile de déterminer combien il y a de classes différentes à séparer.

Les techniques de segmentation sont encore trop peu efficaces pour des images complexes. La segmentation se base, en général, sur une seule information telle que l'histogramme ou encore les contrastes, cependant beaucoup d'images ne peuvent être condensées en une de ces informations seulement. La segmentation trouve donc un domaine d'application plus ou moins efficace dans les images que l'on pourrait qualifier d'images simples. Une image simple serait une image ayant peu de classes, ayant des classes séparées bien

distinctement et dont les conditions d'éclairage sont idéales, c'est-à-dire dont l'ombre, la surexposition, etc ne viennent pas perturber l'information contenue dans l'image. Une image simple devrait aussi contenir des couleurs qui se différencient une fois que l'image est transformée en niveaux de gris. En effet une certaine teinte de rouge peut donner un niveau de gris identique à une certaine teinte de bleu ou encore de vert. Or il est clair qu'à l'œil nu ce sont des objets de couleurs différentes qui devraient faire partie de classes différentes, mais une fois l'histogramme des niveaux de gris créé, ces différences ne sont pas distinguables. Le spectre des couleurs devrait pouvoir être pris en compte dans la segmentation. Parfois il faudra appliquer certains traitements aux images avant d'effectuer une opération de segmentation.

Finalement, une bonne méthode de segmentation fournira un résultat qui serait une image présentant l'information de façon simplifiée sans retirer d'informations essentielles à l'interprétation de ce résultat.

Illustration d'un cas, à la figure 23, où la segmentation s'applique idéalement :
Le seuil a été déterminé à l'aide du seuillage d'entropie maximum (97). Le résultat est presque similaire à l'aide d'Otsu (94), de isoData (91) ou de Mixture Modeling (86).

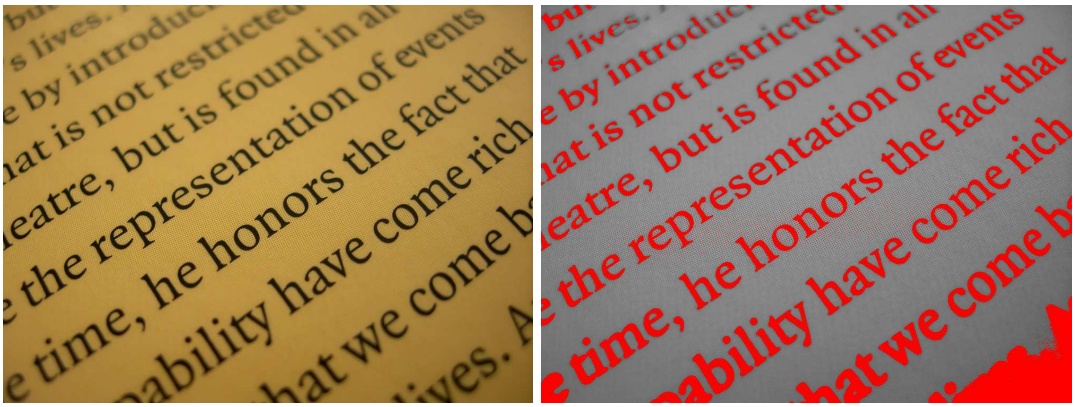

Fig. 23 – Illustration de segmentation idéale

2.5 Homogénéisation d'image

Ce qui sera appelé, tout au long de ce mémoire, l'homogénéisation d'image, vise à améliorer la qualité d'une image en tentant d'homogénéiser la luminosité répartie sur cette dernière. C'est-à-dire que le but de l'homogénéisation est de répartir de manière uniforme la luminosité de l'image. En effet, la luminosité de l'image peut varier selon le point de vue duquel elle a été prise. Prenons le cas de la précédente illustration avec la feuille écrite (la figure 23). Si l'on s'intéresse à la partie inférieure droite, visible à la figure 24, on constate qu'il y a une zone plus sombre et cela se remarque fort sur l'image seuillée car cette zone plus sombre est considérée comme faisant partie de l'écriture (partie colorée en rouge) alors qu'elle fait partie du papier sur lequel il y a l'écriture.

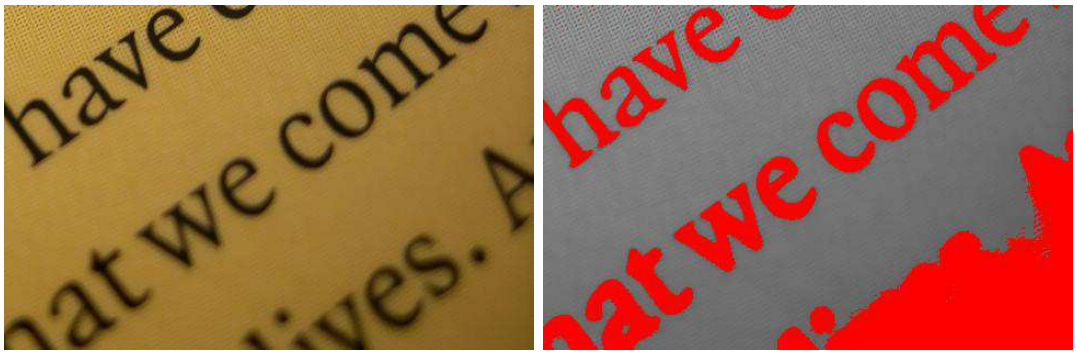

FIG. 24 – Zone sombre (en bas à droite)

2.5.1 *Homogénéisation manuelle*

Si l'on désire homogénéiser cette image, on constate à lœil nu que la partie inférieure droite est plus sombre que la partie supérieure gauche. On en déduit donc qu'il faut éclaircir la partie inférieure droite ou alors assombrir la partie supérieure gauche. On se doute aussi qu'il ne faudra pas éclaircir ou assombrir l'entièreté de l'image avec la même intensité. Et donc, certaines zones devront êtres plus éclaircies ou assombries que d'autres. En quelques sortes on devra éclaircir ou assombrir de façon dégradée.

Finalement, l'homogénéisation de cette image peut se faire manuellement à l'aide d'un programme de traitement d'image. Ci-dessous, voici une illustration de ce que l'homogénéisation manuelle peut donner. La première image est l'image d'origine. En dessous de cette dernière, à gauche, l'image correspond à une homogénéisation en éclaircissant les zones plus sombres. A droite, l'image correspond à un assombrissement des zones plus claires. Ces deux exemples d'homogénéisation ont été faits à l'aide de Adobe Photoshop CS2, en utilisant la fonctionnalité du réglage des niveaux et un dégradé dans le masque de fusion.

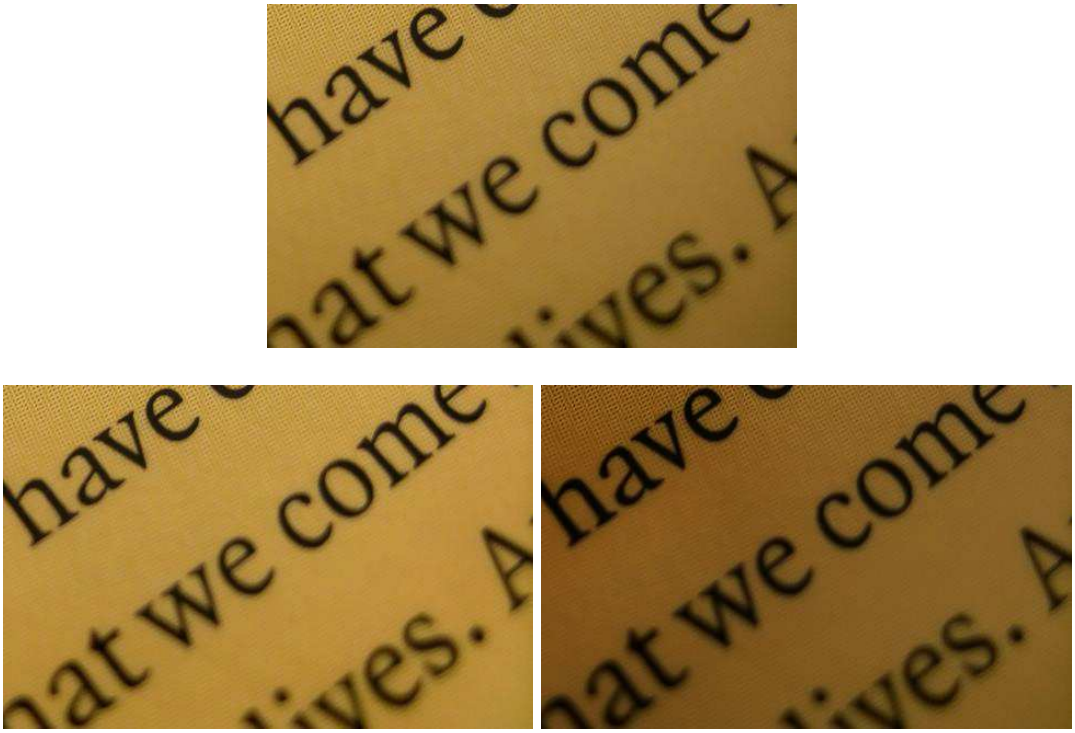

FIG. 25 – Exemple d'homogénéisation manuelle

2.5.2 *Homogénéisation automatique*

Le problème qui se pose ici est de pouvoir homogénéiser des images de façon automatique, sans aucun jugement humain.

Le codage HSL :

L'homogénéisation telle qu'elle a été décrite se base sur la luminosité de l'image. La luminosité doit donc pouvoir être déterminée de façon automatique. Pour ce faire nous allons utiliser le codage HSL qui, tout comme le codage RGB, permet de décrire un pixel de l'image. HSL (en français TSL) provient de **H**ue (teinte), **S**aturation (saturation), **L**uminance parfois aussi nommé **L**ightness ou encore **L**uminosity (luminosité ou encore brillance). La teinte correspond à la couleur du pixel, la saturation va décrire la ternissure de cette couleur (le caractère terne ou éclatant comme par exemple un linge délavé ou alors neuf) et la luminosité traduira l'éclaircissement ou l'assombrissement de la couleur. Ces trois composantes du codage prennent chacune leur valeur entre 0 et 1, par exemple le pixel tout en haut à gauche de l'image d'origine de l'exemple précédent possède une valeur de luminosité de 0.44901960784313727. Un tel codage se justifie par le fait que, par exemple, pour assombrir une couleur à l'aide du codage RGB, il faut diminuer les trois composantes R, G et B dans une même proportion. Ainsi, à l'aide du codage HSL, il suffit de modifier la seule composante L pour arriver au même résultat. C'est pourquoi ce codage va servir

à l'homogénéisation, du fait qu'il nous permet de déterminer la luminosité d'un pixel.

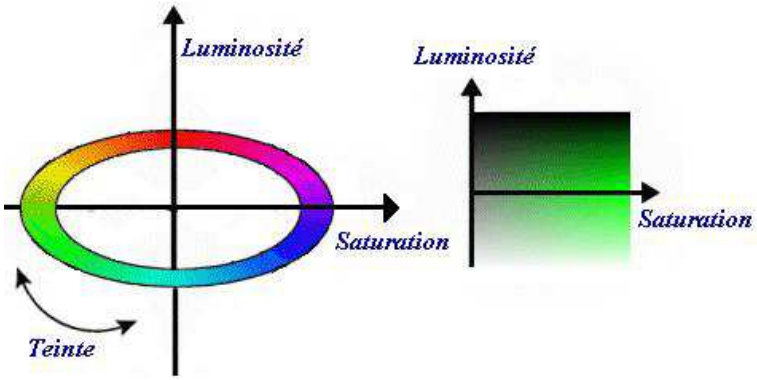

FIG. 26 – Système de codage HSL

L'algorithme de conversion RGB vers HSL et celui de conversion HSL vers RGB, codés en Java, se trouvent en annexe E.

Techniques d'homogénéisation automatique :

Maintenant que nous savons comment obtenir la luminosité d'un pixel nous pouvons tenter d'homogénéiser la répartition de la luminosité sur toute l'image. Une première idée est de corriger la pente qui existe entre une extrémité de l'image et l'autre extrémité. Ainsi, si le pixel au-dessus à gauche possède une luminosité de 0.5, et le pixel en dessous à gauche possède une luminosité de 1, on sait dire que la différence entre les deux extrémités de l'image est de 0.5. Cette différence est donc à corriger en calculant la pente et en redressant cette pente. Bien entendu, il existe plusieurs pentes à corriger. Ci-dessus, la pente dont il est question est la pente sur la hauteur de l'image, mais la pente peut aussi être prise sur la longueur ou encore sur la diagonale. Voici un exemple concret de correction de pente sur la hauteur d'une image de 6 pixels de haut :
Si la luminosité du pixel [0,0] est de 0.5 et la luminosité du pixel [0,5] est de 0.75, la pente de luminosité vaut :

$$pente = \frac{0.75 - 0.5}{5 - 0} = 0.05$$

Ainsi, sachant que la valeur de la luminosité augmente de 0.05 à chaque pixel, la correction peut être effectuée en retirant la bonne valeur de luminosité à chaque pixel.

Le problème de cette méthode c'est qu'elle présuppose que la différence de luminosité d'un pixel à l'autre est linéaire. Or ce n'est pas toujours le cas, en effet, cette différence peut varier d'un pixel à l'autre de façon non prédictible. Ainsi, la luminosité peut augmenter comparativement au pixel suivant et diminuer comparativement à celui encore plus

loin. Ceci donne un effet de montagne russe alors que la pente ne verrait qu'un effet linéaire.

Une seconde méthode de correction consiste à prendre une valeur de luminosité de référence et à s'en approcher le plus possible pour chaque pixel. La valeur de référence considérée ici sera la moyenne de luminosité de toute l'image.

Illustration : ci-dessous, notre image originale avec la zone sombre dans son coin bas-droit. Ensuite, l'image homogénéisée avec la technique de la valeur de référence à approcher. La valeur de référence ici est la moyenne de luminosité.

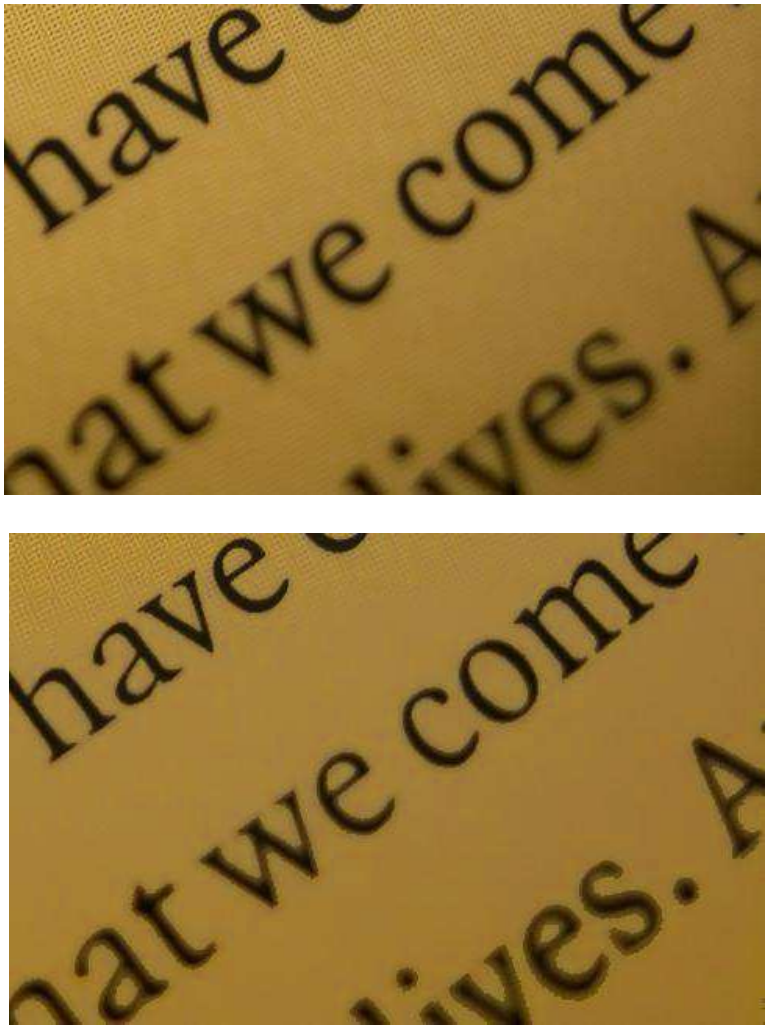

FIG. 27 – Exemple d'homogénéisation automatique

2.6 Communication et exploitation d'images

Dans le milieu hospitalier, les images doivent pouvoir être communiquées à diverses personnes et visualisées par ces personnes. C'est là qu'intervient **DICOM** (Digital Imaging et Communications in Medicine). DICOM est un standard utilisé dans le monde hospitalier (hôpitaux, cliniques, centres spécialisés, etc...) pour la gestion des images (produire, stocker, afficher, envoyer, imprimer, etc...). Ce standard[18] est très utilisé dans le domaine de l'imagerie médicale et pour l'archivage et la communication de telles images. Le format DICOM pour les images est composé d'une multitude de champs décrivant l'image encodée. Chaque champ est accompagné d'un tag constitué d'un numéro de groupe et d'un numéro d'élément. Bien entendu, un des champs contiendra l'image. Voici quelques exemples de champs contenus dans le format DICOM avec une valeur placée dans ces champs :

Champ	**Tag**	**Valeur**
Scheduled Procedure Step Location	(0040,0011)	Laboratoire hématologie
Institution Name	(0008,0080)	UCL Mont-Godinne
Station Name	(0008,1010)	Microscope Virtuel

Fig. 28 – Exemples de champs DICOM

Une possibilité pour visualiser des images au format DICOM est Telemis. Telemis S.A. est une société de Louvain-La-Neuve créée en avril 1999 et compétente dans la conception de solutions pour la gestion d'images médicales.

[18]Description complète du standard sur ce site : `http://www.dclunie.com/dicom-status/status.html`

Troisième partie

p

Résultats

3 Troisième partie : résultats

3.1 Introduction

On se souvient que la première partie abordait le cadre et les objectifs du travail. Précédemment, dans la seconde partie, nous avons vu diverses techniques de traitement d'image. Cette troisième partie va illustrer l'utilisation de ces techniques pour réaliser les objectifs.

Cette partie s'articule autour des trois objectifs principaux qui ont été énoncés dans la première partie :

1. la composition automatique d'une galerie d'images à partir d'un frotti numérisé (point 3.3) ;
2. l'amélioration de la qualité des images par homogénéisation (point 3.4) ;
3. l'extraction d'images contenues dans une base de données (point 3.5) ;
4. la classification des globules blancs (point 3.6) ;
5. le problème de l'autofocus (point 3.7).

Mais avant tout commençons par décrire le langage Java qui a permis la réalisation des objectifs.

3.2 Le langage Java

Le langage utilisé pour la mise en oeuvre des diverses techniques décrites est le langage **Java™**. L'existant ayant été créé à l'aide de Java[19], ses améliorations se font donc dans la continuité et conserve ce langage. Ceci n'était pas pour me déplaire car Java est le langage qui m'a accompagné au long de mes études et dont j'apprécie beaucoup l'utilisation.

Ce langage orienté objet est compilé et interprété. La machine virtuelle, plus connue sous le nom de JVM (Java virtual Machine), interprète le code compilé. Le JDK (Java Development Kit) fournit à la fois un compilateur, une JVM et des bibliothèques d'outils facilitant la programmation en permettant de ne pas réinventer la roue. Ceci offre au langage Java l'avantage de la portabilité. En effet, le code compilé est interprété par la machine virtuelle de l'environnement sur lequel l'application est déployée, il n'y a donc pas de nécessité de recompiler le code source à chaque changement d'environnement. Si l'on se réfère au point 1.4.3, il est question de multitudes d'environnements différents sur lesquels doit pouvoir fonctionner le client de visualisation. Cela sous-entend que l'application doit être portable sur divers environnements. Le langage Java, de par sa portabilité, permet de rencontrer cet objectif.

Le fait que ce soit un langage orienté objet est très utile dans le cadre de notre application. En effet, cette dernière se voit améliorée et dotée de nouvelles fonctionnalités au fur et à mesure que des stagiaires y apportent leur contribution. Il est donc nécessaire de pouvoir

[19] La version de Java, utilisée lors du stage, est la la version 1.4.

posséder une technologie permettant l'ajout de nouveaux modules de façon aisée. De plus comme il a été signalé auparavant, ce langage offre des bibliothèques d'objets utilisables par le programmeur. Ceci représente un avantage indéniable étant donné que Java offre des classes utilisables de façon transparente telles que *Thread*, *Socket*, *Vector* ou encore *Image* et *BufferedImage* pour l'utilisation d'images.

Les avantages de Java peuvent se résumer comme suit :
- c'est un langage puissant et plutôt simple d'utilisation ;
- il génère des applications portables ;
- il permet de créer des interfaces graphiques aisément ;
- son API (Application Program Interface) est très riche ;
- il incorpore une gestion automatique de la mémoire.

Cependant ces avantages sont à relativiser. Je pense notamment à la gestion automatique de la mémoire. Le *Garbage Collector*[20] (GC) permet au programmeur de ne pas se soucier de la libération de la mémoire allouée, ce qui est un avantage. Ceci peut aussi poser problème, en effet, le programmeur n'a pas le contrôle total de la désallocation de la mémoire, il se peut donc que le GC altère les performances d'une application. Le GC pourrait faire un sujet de mémoire à lui seul, mis à part quelques conseils pour l'utilisation de la mémoire en Java (voir 4.3.1), nous ne nous étalerons pas d'avantage sur le sujet.

Le choix de Java pour le codage des applications ainsi que le choix de JPEG2000 comme standard de compression des images ont conduit tout naturellement à l'adoption par L. Zuyderhoff de JJ2000. En effet, JJ2000 est l'implémentation de référence en Java du standard JPEG2000. Cette implémentation est le résultat de la collaboration entre l'EPFL (École Polytechnique Fédérale de Lausanne), Ericsson et Canon.

3.3 La composition automatique d'une galerie d'images de globules blancs

B. Georges proposait, dans ses travaux (voir [Geo02]), un outil, intégré au logiciel AnalySIS, permettant de capturer une galerie d'images des régions d'intérêt d'un frotti. Suite à cela (voir [Zuy03]), le microscope virtuel a permis de capturer une zone significative d'un frotti au format JPEG2000. De façon analogue à l'interaction du microscope réel et de l'outil de création de galerie intégré dans AnalySIS, il est judicieux de pouvoir, à partir du microscope virtuel, créer une galerie des régions d'intérêt. Ceci constitue le premier et principal objectif du travail réalisé durant le stage au laboratoire de Mont-Godinne.

Possédant une zone numérisée d'un frotti sanguin, visualisable via le client de visualisation du microscope virtuel, et sachant que les régions d'intérêt dans notre cas sont les globules blancs de ce frotti, il faut pouvoir recréer une galerie d'images représentant ces globules blancs. A l'œil nu, ces globules blancs sont facilement repérables car de couleur pourpre. Cependant, les couleurs sont modifiables directement à la source, c'est-à-dire sur

[20] Parfois aussi appelé *Garbage Collection*

le microscope réel. Il n'est donc pas garanti que les globules blancs seront toujours de couleur pourpre. De plus, chaque classe de globules blancs ne possède pas exactement la même couleur. Il n'est donc pas avantageux de se baser sur le critère de la couleur puisque celle-ci n'est pas fixe.

Comment permettre alors au programme de détecter de façon automatique les divers globules blancs présents dans la zone numérisée du frotti ? La réponse se trouve dans l'histogramme de l'image. En effet, quelle que soit la teinte de couleur de l'image, en observant l'histogramme, il est possible de distinguer trois classes. Une classe correspondant à l'arrière-plan qui n'est autre que le plasma, une classe qui correspond aux globules rouges et enfin une classe représentant les globules blancs. De cette manière, il est possible de savoir à quelles valeurs de pixel correspondent les globules blancs. Cette étape est l'étape du seuillage. Suite à ça, il faut détecter ces globules blancs dans la méga-image. Cependant il est préférable au préalable de filtrer l'image de façon à éliminer certaines zones considérées comme du bruit. Une fois la détection des globules effectuée en combinant un opérateur de détection de contour et le seuil obtenu lors de la première étape, l'image est seuillée de façon à n'avoir qu'une couleur pour les globules blancs et une couleur pour le reste. Ainsi seuls les globules blancs (et d'éventuels résidus bruités) sont visibles sur un fond de couleur uniforme. Cependant cette image binaire n'est encore qu'une matrice de pixels. Un chaînage des pixels représentant les globules blancs est donc intéressant pour pouvoir obtenir des structures nous en apprenant plus sur la localisation des globules dans l'image, leur taille et bien d'autres informations encore. Les chaînes obtenues, la localisation des globules est donc connue, il est ainsi possible de les extraire de la méga-image. Avec ces images de globules extraites, il est maintenant possible de créer une galerie d'images visualisable via l'interface homme-machine (IHM) du client de visualisation du microscope virtuel. Voici étape par étape la description de la composition d'une galerie à partir d'une méga-image. En résumé, les étapes amenant à la création d'une galerie sont les suivantes :

- le seuillage ;
- le filtrage ;
- la détection de contours ;
- la binarisation ;
- le chaînage ;
- l'extraction ;
- l'affichage dans l'IHM.

3.3.1 *Seuillage*

La première étape de la composition d'une galerie d'images de globules blancs est l'étape de seuillage. Effectivement, si on se souvient, par exemple, de la méthode de seuillage d'Otsu (voir 2.4.1), il est possible de diviser une image en deux classes qui représentent typiquement l'arrière plan et les objets. Dans notre cas, les objets sont les globules blancs, les globules rouges et les plaquettes. L'arrière plan est matérialisé par le plasma. Voici une illustration (figure 29) représentant bien l'idée : nous avons une image d'une zone d'un frotti sanguin vu au microscope. On distingue aisément les globules rouges, les globules

blancs (au nombre de trois) et le fond. A côté l'histogramme associé à cette image. Dans ce dernier, le pic à droite correspond à la partie la plus claire de l'image, c'est-à-dire le fond. Le pic directement à sa gauche correspond à une partie un peu plus foncée, il s'agit de la classe des globules rouges. Et encore plus à gauche, quasiment au centre de l'axe des X de cet histogramme, se trouve une dernière petite protubérance qui représente les globules blancs.

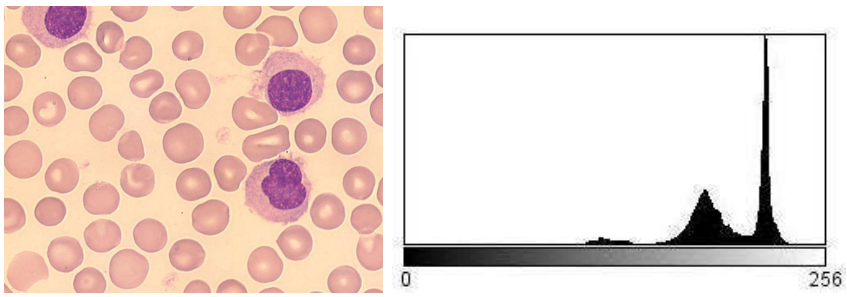

FIG. 29 – Exemple d'histogramme d'un frotti sanguin

Comme il a été dit, une image d'un frotti sanguin peu avoir des couleurs qui changent par rapport à d'autres images de frotti sanguin. Il n'est donc pas possible de généraliser à toutes les images le fait que les globules blancs sont d'une couleur bien déterminée. En utilisant l'histogramme, il est possible de s'adapter à chacune des images, toutes différentes les unes des autres. La figure 29 illustre bien le fait qu'on peut détecter trois classes différentes dans une image du type frotti sanguin. Si les couleurs de l'image changent, ces trois classes seront toujours présentes dans le même ordre mais de façon décalée vers la gauche ou vers la droite, ou encore plus séparées les unes des autres voir plus rapprochées les unes des autres. En utilisant des techniques de seuillage, on peut déterminer un seuil séparant ces diverses classes. Une fois les seuils obtenus, on est en possession de l'information du niveau de gris séparant les sortes de globules (on se souvient que les techniques de seuillage sont pour la majorité adaptées à un histogramme du niveau de gris (voir 2.4). Prenons l'histogramme de la figure 29, le seuil séparant les globules blancs des globules rouges se situe aux alentours du niveau de gris de valeur 160. De même, celui séparant les globules rouges du fond est approximativement autour de la valeur 200. Donc dans notre cas, les pixels ayant une valeur de gris inférieure à 160 sont des pixels appartenant aux globules blancs, les pixels situés entre 160 et 200 sont ceux appartenant aux globules rouges, et enfin, ceux de valeur supérieure à 200 composent le fond. De cette façon, il est donc possible de connaître les pixels appartenant aux diverses classes agencées dans l'image et ce pour chaque image, quelles que soit ses couleurs.

Tout d'abord, il est nécessaire d'obtenir l'histogramme en niveaux de gris de l'image. Pour cela, l'image est transformée en niveaux de gris via la formule "niveau de gris = $(R + G + B)/3$" qui permet d'obtenir le niveau de gris d'un pixel à partir de ses composantes RGB (voir point 2.2). Une fois l'image transformée, il faut composer l'histogramme en comptabilisant le nombre de fois que chaque valeur de niveau de gris se retrouve dans

l'image. Il s'agit alors d'obtenir les seuils séparant les diverses classes présentes dans cet histogramme. Mais quelles méthodes de seuillage utiliser ? Voici un exemple à la figure 30 de ce que produisent la méthode d'Otsu (voir point 2.4.1) et celle de l'entropie maximum (voir point 2.4.2) à partir de l'image donnée à la figure 29.

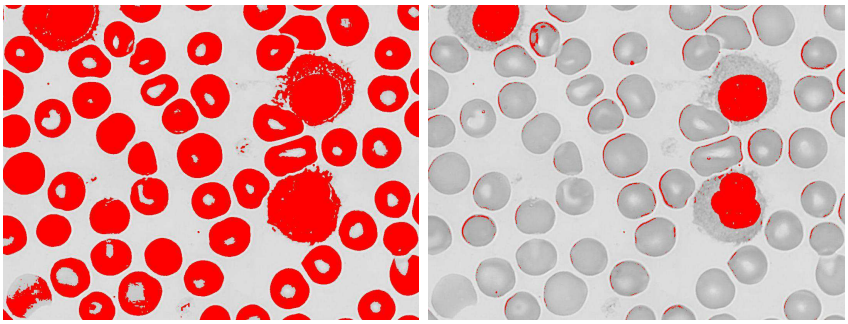

FIG. 30 – Utilisation de la méthode d'Otsu (gauche) et de l'entropie maximum (droite) sur un frotti sanguin

On constate aisément dans cet exemple de la figure 30 que la méthode d'Otsu permet d'obtenir le seuil qui va délimiter les globules du fond. De même, on perçoit que la méthode de l'entropie maximum permet de déterminer le seuil séparant les globules blancs du reste composé des globules rouges et du fond. Notre but étant d'obtenir une galerie des globules blancs, une conclusion hâtive serait d'utiliser directement la méthode de l'entropie maximum. Néanmoins il faut tenir compte de divers cas. Premièrement, certaines images utilisées pour les tests de ces méthodes ne présentaient aucun globule blanc. Deuxièmement, certaines images de tests présentaient des couleurs plus sombres que d'autres et ces images ne permettaient pas à la méthode de l'entropie maximum d'obtenir des résultats concluants. Voici à la figure 31 un exemple où l'entropie maximum donne un résultat non désiré.

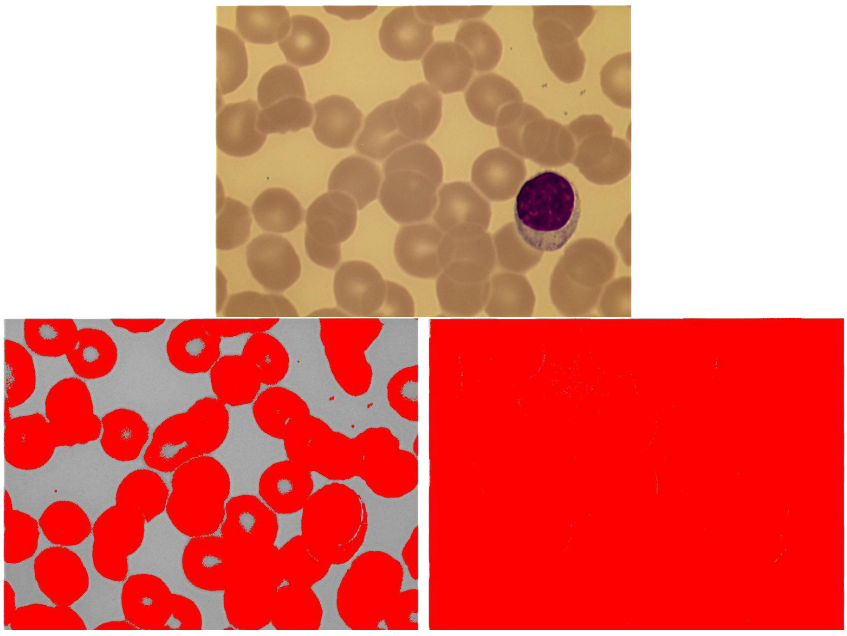

FIG. 31 – A partir de l'image originale (au-dessus), voici un exemple de la méthode d'Otsu (à gauche) et d'un résultat non désiré de l'entropie maximum (à droite)

Dès lors, que faire pour détecter les globules blancs ? La méthode d'Otsu dans chacun des tests effectués retournait le seuil séparant les globules du fond. Donc via cette méthode on sait obtenir le seuil sous lequel se trouvent les globules, qu'ils soient rouges ou blancs. Or dans chaque image de frotti sanguin, il y a des globules. Comme on a pu le constater, la méthode de l'entropie maximum marche dans certains cas pour obtenir le seuil sous lequel se trouvent les globules blancs. Quels sont ces cas ? L'exemple de la figure 30 nous montre un cas favorable. A partir de ce cas, on a remarqué que le seuil obtenu à l'aide de la méthode d'Otsu se situe aux alentours de 200. L'idée développée est de pouvoir ramener chaque cas à ce cas favorable. Pour ce faire, il faut déterminer le seuil d'Otsu. Ensuite il faut modifier l'image de façon à ce que le seuil d'Otsu soit égal à 200. Comment modifier l'image ? Simplement en éclaircissant ou en assombrissant l'image. En effet, par exemple si le seuil d'Otsu obtenu est égal à 170, c'est que l'image est plus foncée qu'espéré dans le cas favorable. Il faut donc l'éclaircir en ajoutant 30 (obtenu en faisant $200 - 170$) à chaque composante RGB de l'ensemble des pixels de l'image. Le seuil retourné par la méthode d'Otsu variera dans les mêmes proportions que la modification de l'image, il sera donc égal à 200, ce qui représente le cas favorable à l'application de la méthode de l'entropie maximum. De même, si l'image est trop claire et nous permet d'obtenir un seuil d'Otsu supérieur à 200, l'image sera assombrie de façon à atteindre un cas favorable. Illustration à la figure 32 : repartons de l'image de la figure 31 qui était un cas défavorable comme on l'a vu, appliquons la méthode d'Otsu, le seuil obtenu est égal à 143. Ce seuil est beaucoup plus bas qu'espéré par le cas favorable qui est un seuil de 200. Modifions l'image en l'éclaircissant afin de parvenir à un cas favorable. Pour l'éclaircir, on ajoute la valeur 57 à chaque composante RGB de chaque pixel. Une fois fait, l'image est devenue favorable, en effet, l'éclaircissement de cette dernière nous donne un seuil d'Otsu de 200. On peut donc appliquer la technique de l'entropie maximum et le seuil obtenu via cette technique est celui sous lequel se trouvent les globules blancs, plus précisément le globule blanc dans notre exemple.

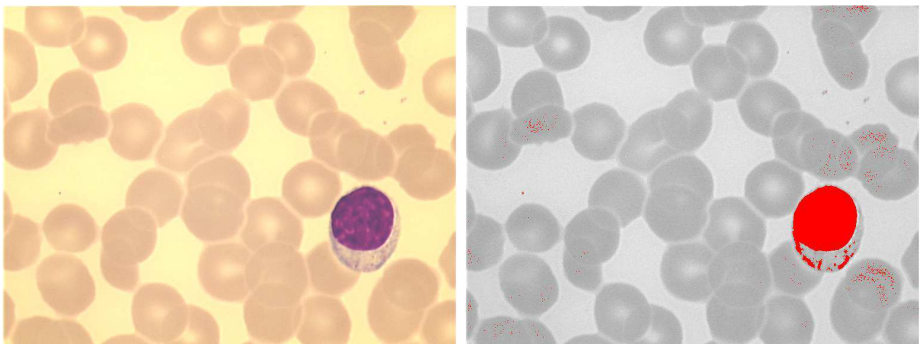

FIG. 32 – Modification de l'image pour la réussite de l'entropie maximum

Nous connaissons maintenant le seuil séparant les globules blancs du reste. Tous les pixels de l'image se trouvant au-delà de ce seuil n'appartiennent pas aux globules blancs, mais aux globules rouges ou au fond. La méthode développée dans ce travail remplace tous ces pixels par des pixels blancs. L'image obtenue après ça est une image contenant unique-

ment les globules blancs sur un fond blanc. Ceci représente le résultat de la première étape.

Remarque : plus précisément, cette étape de seuillage nous permet de détecter les noyaux des globules blancs et non leur cytoplasme[21]. En effet, le cytoplasme entourant le noyau du globule blanc est d'une couleur se rapprochant trop des globules rouges que pour être repris dans une autre classe de l'histogramme que celle des globules rouges. Par abus de langage, nous parlerons de détection de globules blancs alors qu'en réalité il s'agit de la détection des noyaux des globules blancs.

3.3.2 *Filtrage*

La seconde étape consiste en un filtrage de l'image obtenue à la première étape qui est celle de seuillage. Pourquoi filtrer cette image ? Simplement parce que le seuillage peut ne pas faire ressortir uniquement les globules blancs. Effectivement, certaines zones, que nous appellerons du bruit, peuvent rester dans l'image suite à l'étape de seuillage. C'est le cas par exemple si plusieurs globules rouges se recouvrent partiellement, alors l'intersection de ces recouvrements sera plus foncée que ne l'est un globule rouge seul. On peut aussi rencontrer des zones de couleur plus sombre un peu partout dans l'image. Finalement, ces zones sont de petite taille ce qui permet à l'aide d'un filtrage médian (voir 2.3.2) de supprimer certaines de ces zones. L'exemple de la figure 33 montre un globule blanc (le plus foncé de tous) entouré de globules rouges. On remarque quelques taches de couleur plus foncée que la couleur des globules rouges un peu partout dans l'image (notamment dans le globule rouge directement au-dessus du globule blanc). A côté, l'image illustre ce que le seuillage permet d'obtenir comme résultat. Et finalement, la dernière image correspond à l'application d'un filtre médian sur le résultat du seuillage. Ceci a pour effet de retirer le bruit et de ne retenir que ce qui est pertinent, c'est-à-dire le globule blanc.

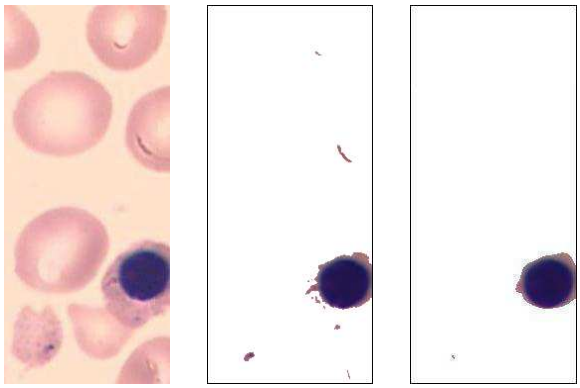

FIG. 33 – Image d'origine → seuillage → filtrage médian pour éliminer le bruit

Bien entendu, il arrive que certaines zones de bruit soient trop grandes que pour disparaître entièrement après le passage d'un filtre médian. Nous verrons plus tard comment considérer ces zones de bruit (voir point 3.3.5). Finalement, le résultat de cette seconde étape est une image ébruitée des globules blancs sur un fond blanc.

[21] Selon Le Petit Larousse Illustré, un cytoplasme est une « *partie fondamentale, homogène, de la cellule, qui contient le noyau, les vacuoles, le chondriome et les autres organites* ».

3.3.3 *Détection*

Une fois l'étape de filtrage médian terminée, il faut passer à une nouvelle étape de filtrage qui est un filtrage pour détecter les contours. Comme expliqué dans le point 2.3.4, les opérateurs de détection de contours se basent sur la luminosité pour détecter des contours de formes. Ils supposent qu'il y a un contour dès qu'on observe une variation flagrante de la luminosité de l'image. Notre image à traiter est une image des globules blancs sur fond blanc. Ces globules blancs marquent inévitablement une discontinuité dans la luminosité par rapport au fond blanc. Dans notre exemple précédent, sur la première ligne de l'image, le premier pixel appartenant au globule blanc possède une valeur de luminosité (selon le codage TSL, voir point 2.5.2) égale à 0.48039215. Ceci marque nettement une interruption dans la luminosité de l'image puisque la luminosité du fond blanc est égale à 1. Il est donc aisé d'obtenir les contours précis des globules blancs avec une telle image.

Dans notre cas, voici comment appliquer, en Java, le filtrage de détection de contours : premièrement, il faut définir le noyau[22] de l'opération de convolution [23], dans notre cas il s'agit d'une matrice 3×3 composée de réels (f signifiant float). Ensuite il faut invoquer, à partir de ce noyau, la fonction *filter()* sur l'image. Le résultat obtenu est l'image filtrée.

```
/* Définition du noyau de l'opération de convolution */
ConvolveOp op = new ConvolveOp(new Kernel(3,3, new float[]{
                        0.0f, -0.75f, 0.0f,
                        -0.75f, 3.65f, -0.75f,
                        0.0f, -0.75f, 0.0f
                }));

/* Application du filtrage par convolution */
BufferedImage imageFiltree = op.filter(imageOriginale,null);
```

Voyons maintenant le résultat obtenu en appliquant ce filtre. La figure 34 nous montre une image de départ (1), suivie de l'image obtenue après le seuillage et le filtrage médian (2). Les deux dernières images illustrent l'application du filtre de détection de contours (3 et 4). On constate que les couleurs de l'image ont changé. Si on zoome dans l'image obtenue (4), on constate que le contour est délimité par une ligne blanche d'un pixel de large côte à côte avec une ligne noire d'un pixel de large.

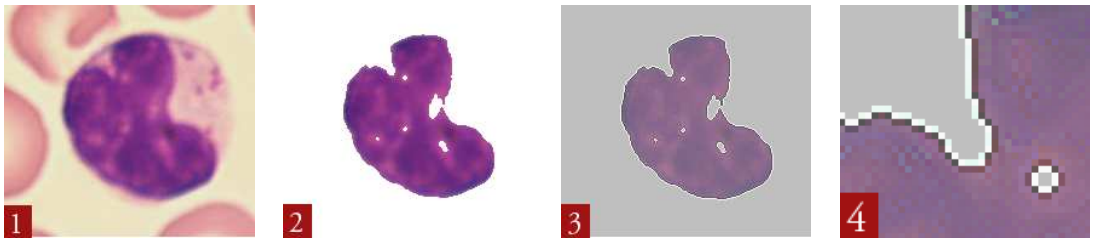

FIG. 34 – Détection des contours (3 et 4) après seuillage (2) de l'image originale (1)

Cette troisième étape nous a permis d'obtenir une image où les globules blancs sont encore présents, mais ces derniers sont bien délimités par un fin contour de couleur blanche.

[22] Ce filtre a été trouvé dans [Fla01], p.355 illustrant des exemples de filtrages d'image en Java.
[23] voir point 2.3.1 pour l'explication du fonctionnement d'un filtrage par convolution.

3.3.4 *Binarisation*

L'étape de détection de contours nous a fourni une image où les contours des globules sont mis en évidence à l'aide d'une ligne blanche. Maintenant, appliquons une opération de binarisation (voir point 2.4.4) de cette image. Le but est d'obtenir une image simplifiée où seules deux valeurs sont possibles. Plus précisément, les deux valeurs qui seront utilisées sont le noir pour les contours des globules et le blanc pour tout le reste. Comment arriver à ce résultat à partir de notre image ? On a pu constater que le contour d'un globule est maintenant mis en évidence via la couleur blanche. Nous avons aussi remarqué que la couleur blanche du fond a cédé sa place à une couleur grise (un niveau de gris de valeur 165 plus précisément). Au vu de ces résultats, on se doute que seuls les contours sont blancs. L'image de la détection de contours sera donc parcourue pixel par pixel et sa couleur sera analysée. Si celle-ci correspond à du blanc alors c'est un contour, et donc ces pixels vont se voir attribuer une couleur noire. Tous les autres pixels seront changés en blanc. Le résultat ainsi obtenu est bel et bien une image à deux valeurs (le blanc et le noir) dont le blanc correspond aux contours des globules blancs qui, rappelons le, sont les éléments que l'on recherche dans la méga-image originale.

Voici le résultat obtenu à la suite de cette étape de binarisation : la figure 35 montre une image originale d'un globule blanc et de quelques globules rouges. A côté, l'image obtenue une fois chaque étape appliquée jusqu'à la binarisation y comprise. Cette dernière ne montre plus que le contour du globule blanc (ou plus précisément de son noyau).

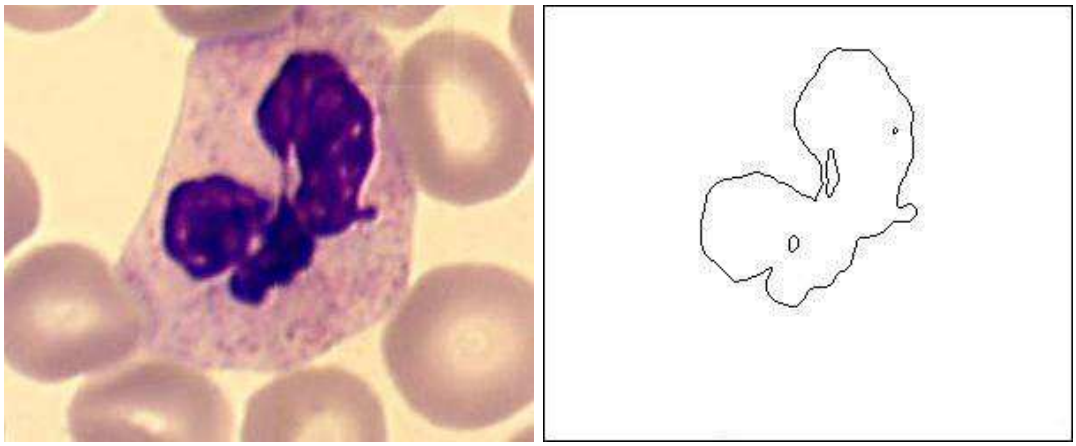

Fig. 35 – Binarisation pour ne faire ressortir que les contours

3.3.5 *Chaînage*

Actuellement, nous avons une image binaire mettant en évidence les contours des noyaux des globules blancs. Cependant une telle image ne reste qu'une simple matrice de pixels. Il faut pouvoir retirer de l'information utile de cette matrice. L'étape de chaînage va permettre de chaîner les contours et chaque chaîne qui représentera un contour

pourra contenir de l'information utile. En effet, une chaîne, dans notre cas, est une structure composée de la liste des pixels la composant, des coordonnées[24] du carré dans lequel le globule est inscrit (ce qui sera utile pour créer une image du globule), de la taille de la chaîne. D'autres informations peuvent être ajoutées aisément à cette structure, ce que nous ferons plus tard.

Pour chaîner les divers contours de l'image, qui sont représentés par les pixels noirs, il faut parcourir toute l'image pixel par pixel. La technique utilisée est proche de la technique dite du chaînage par automate (voir point 2.3.4). En effet, chaque pixel noir rencontré est placé dans une chaîne et on tente d'extraire le reste du contour pour l'ajouter à la chaîne. Cependant, il est possible de ne pas pouvoir extraire la chaîne dans son entièreté. En effet, si un embranchement est rencontré, il faut bien continuer par un des chemins du contour et laisser l'autre en suspens. La phase de chaînage est donc suivie d'une phase de fusion des chaînes adjacentes. Cette fusion est possible en comparant les coordonnées des éléments des chaînes, et dès que deux éléments, chacun d'une chaîne différente, sont de coordonnées adjacentes, une fusion est possible. Dès qu'un pixel noir est inscrit dans une chaîne, il se voit attribuer une couleur blanche de façon à ce qu'il ne soit plus jamais pris en compte par le processus de chaînage. On s'assure ainsi qu'un pixel fait partie d'une et une seule chaîne.

Si l'on se remémore la seconde étape qui consistait à filtrer l'image avec un filtre médian de façon à éliminer le bruit, on y constatait que certaines zones, dites "de bruit", n'étaient pas effacées entièrement. L'étape de chaînage ici présente nous apporte une possibilité de supprimer ces zones de bruit. Si, à l'issue de l'étape de chaînage, il existe des chaînes de petite taille, comme par exemple une chaîne de 10 pixels de long, alors on peut la considérer comme étant du bruit et donc la supprimer. Le piège dans lequel il ne faut pas tomber est de supprimer des chaînes utiles. Il faut donc pouvoir déterminer une taille limite sous laquelle les chaînes peuvent être supprimées. Cette taille n'a pas encore été déterminée et fait donc partie des perspectives d'amélioration (voir point 4.4.1).

Les chaînes, représentant chacune un globule blanc, sont des structures contenant des informations utiles. Dans le cadre de la création d'un galerie d'images des globules blancs, les informations les plus utiles sont les coordonnées des globules blancs dans la méga-image d'un frotti sanguin. Cependant d'autres informations peuvent être intéressantes pour trier ces globules en différentes classes, ce qui sera abordé au point 4.4.1. Une des informations utiles est de connaître la surface de ces globules pour pouvoir les comparer. Ici, l'information de surface retenue est celle du nombre de pixels composant un globule. Dès lors pour faciliter le calcul de cette information, il n'est plus nécessaire de passer par l'étape de détection de contours et de binarisation. En effet, l'image résultat de l'étape de filtrage par filtre médian est une image contenant les globules blancs sur un fond blanc. Le chaînage n'a plus qu'à prendre en compte tous les pixels qui ne sont pas blancs. Chaque chaîne ne représentera plus seulement un contour, mais l'entièreté d'un globule blanc (ou plus

[24]Les coordonnées se présentent sous la forme (x, y) où x est la colonne du pixel dans la méga-image et y la ligne de ce même pixel dans cette même méga-image.

précisément d'un noyau d'un tel globule). Tous les pixels d'un globule seront chaînés entre eux et au final, la longueur de la chaîne correspondra au nombre de pixels composant ce globule. Et voilà, l'information de la surface en nombre de pixels est maintenant connue. Mais revenons à ce qui nous intéresse, le but ici est de créer une galerie d'images et non de trier ces images. Le chaînage des contours nous apporte suffisamment d'informations pour extraire les images des globules blancs puisqu'on connaît leurs coordonnées dans la méga-image. Nous pouvons passer à l'étape suivante.

3.3.6 *Extraction*

L'étape du chaînage nous permet de connaître les coordonnées dans la méga-image pour chaque globule blanc. Connaissant ces coordonnées, il faut maintenant extraire des images pour composer la galerie d'images des globules blancs. Les coordonnées contenues dans une chaîne représentant un globule blanc représentent les quatre coins du carré dans lequel le globule (plus précisément le noyau) est inscrit. Pour ne pas créer une image uniquement resserrée autour du globule blanc, il faut élargir un peu ces coordonnées de façon à avoir une image correcte présentant le voisinage du globule en question. Il faut finalement découper de petites images dans la méga-image.

La particularité de cette étape réside dans le fait qu'il n'est pas nécessaire de parcourir l'entièreté de la méga-image pour recréer les images des globules qui nous intéressent. En effet, une des particularités de JPEG2000, qui est le format dans lequel sont encodées les méga-images, est de pouvoir accéder à n'importe quelle "*tile*" souhaitée. Une tile est un morceau d'une image JPEG2000[25]. Ce système de tiles permet l'accès aléatoire à des parties d'une image JPEG2000. Ainsi, connaissant les coordonnées d'une image d'un globule blanc, il est avantageux de demander à JJ2000 les tiles qui nous intéressent pour composer cette image. Par exemple, supposons une image JPEG2000 de 100 pixels sur 140 pixels composée de tiles de 20 pixels sur 20 pixels. L'image est donc composée de 35 tiles réparties en 5 colonnes et 7 lignes. Si, dans cette image, on détecte un globule dont l'image qu'on souhaite en faire possède la coordonnée (52, 63) pour son coin supérieur gauche, on peut en déduire que la tile contenant le pixel de cette coordonnée est la tile située à la troisième colonne de la quatrième ligne.

Les images qui vont composer la galerie sont donc extraites en demandant les tiles qui nous intéressent. Ceci offre un gain de temps et d'espace mémoire puisqu'il ne faut charger que les tiles réquisitionnées et non l'image entière. Quand on sait que les méga-images sont de grande taille et qu'on souhaite appliquer cette création de galerie à des images de plus grande taille que ce que CellaVision peut supporter, cette technique des tiles est très avantageuse.

[25] Chaque image JPEG2000 est composée de sous-éléments appelés tiles ("tuiles" en français). « *Le terme "tiling" désigne la partition d'une image source en blocs rectangulaires disjoints et codés indépendamment (un peu comme s'ils étaient des images distinctes)* »[Zuy03]

3.3.7 *L'IHM*

Nous sommes maintenant en possession des diverses images composant la galerie d'images. Il ne reste plus qu'à les afficher. Ce point présente l'IHM permettant l'affichage de cette galerie et les particularités de cette IHM.

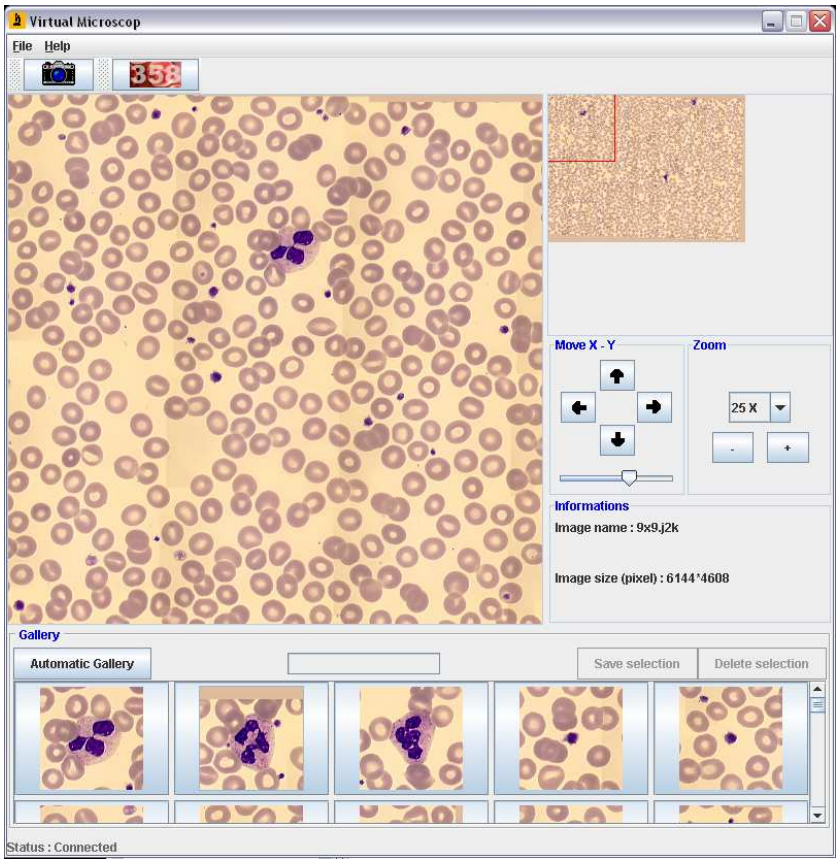

FIG. 36 – IHM de la galerie d'images

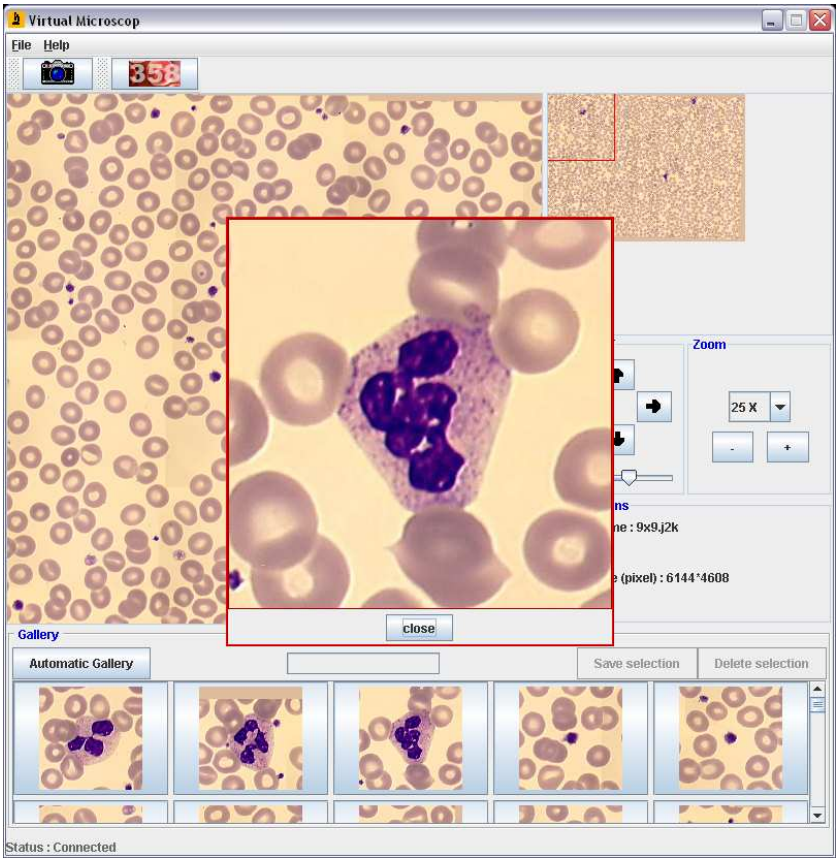

FIG. 37 – IHM de la galerie d'images

La figure 36 illustre le client de visualisation du microscope virtuel. Si l'on se réfère à la figure 6 du point 1.4.2, on constate que l'interface graphique a peu changé. Elle se voit ajouter une zone, dans le bas de ce qui était déjà existant, permettant de lancer la création automatique d'une galerie d'images (bouton "*Automatic gallery*"). Une barre de progression est présente, permettant de voir la progression dans l'avancement de la création de la galerie. Cette progression est divisée en quatre étapes : "*Thresholding*", "*Detecting*", "*Creating*", "*Displaying*". Ces quatre étapes représentent l'étape de seuillage, de détection des contours jusqu'au chaînage, de création des images et finalement d'affichage de ces images dans l'IHM. Une fois les images de la galerie affichées, il est possible de sélectionner les images selon notre bon vouloir. Les images sont affichées dans un "*JToggleButton*" pour pouvoir les sélectionner ou les désélectionner. Les images sélectionnées peuvent soit être supprimées de la galerie (bouton "*Delete selection*" à droite) soit être enregistrées sur le disque (bouton "*Save selection*"). L'enregistrement de la sélection sur le disque se fait via la classe *JFileChooser* (se référer à l'annexe I.1 pour modifier un *JFileChooser* à notre guise). Ce *JFileChooser* permet de choisir l'endroit sur le disque où l'on souhaite sauvegarder la sélection et permet de donner un nom à cette sélection. Chaque image de la sélection portera ce nom suffixé par un numéro d'ordre et sera sauvegardée au format

JPEG puisqu'il ne s'agit plus ici que d'images de petite taille. L'affichage des images dans la galerie se fait à l'aide d'un *GridLayout*[26] appliqué à un *JScrollPane*.

Remarque : la création d'une galerie détecte aussi les plaquettes contenues dans le sang en plus des globules blancs. Ceci peut être observé dans la galerie des figures précédentes (la figure 36 ou la figure 37) où les trois premières images de la galerie, en bas dans l'IHM, sont centrées sur des globules blancs et les suivantes sur des plaquettes. Ceci vient du fait que la couleur des plaquettes est similaire à celle du noyau des globules blancs. Cependant, la détection des plaquettes peut être intéressante étant donné qu'elles peuvent être prises en compte dans l'établissement d'un diagnostic. Le cas échéant, ces dernières pourront être ignorées durant la création en précisant une taille sous laquelle les détections ne seront pas prises en compte. En effet, les plaquettes sont plus petites que les globules blancs.

Une autre fonctionnalité ajoutée à la galerie d'images est la possibilité de visualiser une image de la galerie en pleine résolution, simplement en double cliquant sur l'image souhaitée (voir figure 36). Cette image s'affiche dans une *JWindow* qui est une fenêtre minimaliste puisque réduite à ses quatre bords. Cette fenêtre peut être liée à la fenêtre principale du microscope virtuel en passant cette dernière en paramètre du constructeur de la *JWindow*.

Au passage, on peut remarquer que la zone prévue pour afficher de l'information, et jusqu'alors inutilisée, se remplit des informations concernant la méga-image. En l'occurrence les informations affichées sont le nom de la méga-image et la taille de cette dernière en pixels. Un autre ajout graphique réside dans l'icône de la fenêtre représentant un microscope sur fond orange et créée à partir de rien (voir la figure 53 en annexe pour voir cette image en pleine résolution).

3.3.8 *Spécificités de la création d'une galerie*

Un problème rencontré lors du développement de la création de galeries est le problème de la vitesse d'exécution. En effet, les méga-images étant de grande taille, les divers traitements à y appliquer prennent un certain temps. Pour réduire de façon significative ce temps nécessaire à la création d'une galerie, nous pouvons tirer parti du format JPEG2000 dans lequel les méga-images sont encodées. Comme il a été expliqué précédemment, JPEG2000 permet d'accéder à des parties d'images via un système de tiles. La première étape qui correspond au seuillage est composée du calcul de deux seuils, celui d'Otsu et celui de l'entropie maximum. Pour réduire le temps de calcul de ces deux seuils, on peut tout à fait les calculer sur un sous-ensemble de la méga-image et non sur son entièreté. Arbitrairement nous avons choisi de prendre 36 tiles. Si la méga-image ne possède pas autant de tiles alors l'étape de seuillage s'effectuera sur l'entièreté de la méga-image. Sachant qu'une tile en pleine résolution est généralement de dimension 256 pixels sur 256 pixels voire plus, notre échantillon possède donc plus de 2 millions de pixels pour déterminer les divers seuils

[26] Pour afficher 5 images sur une ligne, *GridLayout* doit être construit comme suit : *GridLayout grille = new GridLayout(0,5,5,5) ;* et initialisé comme suit : *grille.setColumns(5) ;*.

nécessaires.

Une deuxième idée tirant parti du format JPEG2000 est d'utiliser les diverses résolutions que ce format nous autorise à obtenir. En effet, il est possible de demander des tiles dans diverses résolutions. Il existe six résolutions différentes, numérotées de 0 à 5, où 5 est la résolution de l'image à sa taille normale. Les autres résolutions permettent d'obtenir l'image réduite par rapport à sa taille initiale. La figure 36 représentant l'IHM permet de se rendre compte de ce système de résolutions. La méga-image affichée dans le visualisateur est affichée avec la résolution numéro 3 et l'image donnant une vue d'ensemble du frotti (en haut à droite du visualisateur) est cette même méga-image en résolution numéro 0. La résolution numéro 5 n'offre aucune réduction de l'image, cette dernière est donc à 100%. La résolution numéro 4 réduit cette image à 50%. La numéro 3 la réduit à 25%, la 2 à 12%, la 1 à 6% et finalement la 0 à 3%. L'avantage que l'on peut retirer de cette fonctionnalité de JPEG2000 réside dans le fait que l'on peut ainsi faire la détection des globules blancs sur une image réduite afin de localiser les tiles contenant un ou plusieurs de ces globules. Il reste ensuite à récupérer ces tiles en pleine résolution pour continuer le processus de création d'une galerie. La tâche se complique lorsqu'un globule blanc se trouve à l'intersection de plusieurs tiles. Dans ce cas il faut ajouter une opération de détection des tiles adjacentes afin de les regrouper en une image cohérente. En effet, ce serait dommage d'afficher une galerie de morceaux de globules, ceci s'apparenterait à un puzzle.

Diverses méga-images de diverses tailles ont servi à la création et aux tests de l'application. La plus grande méga-image est une image composée de 20 captures en colonne et 20 captures en ligne. Ce qui donne un total de 400 captures faites au microscope et mises côte à côte à l'aide de la coregistration. Clairement, l'image qui en résulte est une méga-image de 13568 pixels de large sur 9984 pixels de haut, ou encore 53 tiles en largeur sur 39 tiles en hauteur, pour un total de 135462912 pixels. Cette image représente, en largeur, environ 1cm du frotti étalé sur la lame de verre observée par le microscope. Elle peut être vue à la figure 50 en annexe.

Une autre technique utilisée pour l'amélioration de la vitesse de création de la galerie est le multithreading. On vient de voir que la localisation des tiles pertinentes se fait en résolution réduite. Une fois cette localisation effectuée, la suite des étapes se fait en pleine résolution sur chacune de ces tiles. Un thread s'occupe de la localisation et de l'envoi des tiles, un autre lit ces tiles et continue le processus. Un dernier thread s'occupe de lire les globules obtenus en pleine résolution et de les afficher dans la galerie. Ces améliorations ont permis d'améliorer le temps d'exécution en passant, pour l'image 20×20 décrite précédemment, d'environ 35 minutes à environ 9 minutes sur la machine utilisée lors du stage.

Ceci termine la présentation des résultats obtenus pour la création de galeries automatiques à l'aide des outils (filtres, seuillage,...) présentés dans la partie "Matériel et méthode". Une telle création apporte un plus à l'application de visualisation, ouvrant ainsi la voie de l'aide au diagnostic.

3.4 L'homogénéisation d'image

Afin de pouvoir développer et tester la fonctionnalité de création de galerie automatique, diverses méga-images ont dû être acquises. Si l'on se réfère à la figure, une méga-image est composée de diverses images acquises et placées côte à côte par un processus de coregistration mis en place par L. Zuyderhoff. Chaque image acquise, composant une méga-image, ne possède pas une luminosité homogène sur l'ensemble de sa surface. Dès que la méga-image est composée, il est possible de discerner les jonctions entre ces images étant donné la variation de luminosité entre ces images. Si l'on se réfère à la figure 1 qui est la toute première figure de ce mémoire, décrivant les divers éléments du sang, on peut remarquer la jonction de ces images composantes en haut à droite de la figure (d'autres exemples de figures vont suivre).

Dès lors, l'objectif est de pouvoir homogénéiser ces méga-images afin d'en améliorer la qualité. Comme il a été expliqué précédemment au point 2.5, l'homogénéisation peut se faire de façon manuelle à l'aide d'un outil de traitement d'image. L'avantage est que l'utilisateur peut apporter son propre jugement sur les zones qui sont à améliorer, mais ce processus est long puisqu'il faut l'appliquer à chaque image composante d'une méga-image. Ce processus devait donc pouvoir se faire de façon automatique. Diverses techniques imaginées ont été testées sans succès comme, par exemple, calculer et corriger la pente existante entre la luminosité du haut d'une image et celle du bas de cette même image (voir point 2.5.2).

Finalement, l'idée retenue fût celle de corriger la luminosité en tentant de se rapprocher d'une luminosité choisie comme référence. La luminosité de référence choisie dans notre cas est la moyenne de la luminosité des pixels composants toutes les images acquises. Ainsi, toutes les images composant une méga-image sont acquises à l'aide du microscope. Ensuite, la luminosité de chaque pixel de chaque image est calculée. La moyenne de l'ensemble de ces luminosités est calculée et retenue comme luminosité de référence.

Le serveur de création permettant la coregistration a donc été modifié de façon à homogénéiser les méga-images qu'il crée. Une fois que ce serveur a déterminé la luminosité de référence, il modifie les images composantes de façon à se rapprocher de cette référence en chaque pixel. Il est bel et bien question de se rapprocher et non d'égaler car l'utilisateur peut définir un coefficient d'homogénéisation. Ce coefficient prenant sa valeur dans l'ensemble $[0, 1]$, où 0 correspond à la meilleure homogénéisation et 1 à aucune homogénéisation, permet de déterminer la modification de la luminosité lors de l'homogénéisation. Ce coefficient est multiplié par la différence qui existe entre la luminosité du pixel et celle de référence. La valeur ainsi obtenue est ajoutée à la luminosité du pixel concerné. Appliqué à tous les pixels, ceci permet d'homogénéiser l'image. Par exemple, si on a un pixel avec une luminosité de 0.5 et que la luminosité de référence est de 0.9, la différence de luminosité est de 0.4. Si l'utilisateur a décidé d'utiliser un coefficient de 0.3, en multipliant la différence de luminosité par ce coefficient, on obtient $0.4 \times 0.3 = 0.12$. Ensuite, en retirant cette valeur obtenue à la valeur de luminosité de référence, on obtient $0.9 - 0.12 = 0.78$.

Cette valeur est appliquée au pixel courant comme étant sa nouvelle valeur de luminosité. Effectivement, cette nouvelle valeur obtenue se rapproche de la valeur de référence.

Pourquoi utiliser un coefficient d'homogénéisation ? En effet, il suffirait d'appliquer directement la luminosité de référence à chaque pixel. En fait, l'homogénéisation des images a un effet sur la coregistration de ces dernières. Globalement, l'homogénéisation a tendance à améliorer la coregistration des images. Cependant, cette coregistration n'est pas forcément la meilleure avec un coefficient de 0 (ce qui équivaut à appliquer la luminosité de référence à chaque pixel). En effet, nous avons constaté que dans bon nombre de cas, un coefficient de 0.2 permet d'obtenir une meilleure coregistration qu'avec un coefficient de 0.1 ou encore de 0. Souvent dans ces cas là, la coregistration avec un coefficient 0 n'améliorait aucunement la coregistration qui restait la même que si l'image n'avait pas été homogénéisée. Le coefficient est donc justifié par un double avantage, il permet d'homogénéiser l'image et il permet d'améliorer la coregistration. Le tout est de trouver le bon compromis par des tests. Pour faire les divers tests avec divers coefficients, il suffit de recommencer le processus de coregistration à partir des images acquises et sauvegardées sur le disque. Voici un exemple d'homogénéisation et d'amélioration de la coregistration à la figure 38 et à la figure 39. Dans la première méga-image, on remarque la différence de luminosité qui met en évidence la jonction des images. Cette différence se matérialise par un fond qui passe du rose clair à un rose plus foncé (entouré en rouge). Juste à côté se trouve cette même méga-image dont les images composantes ont été homogénéisées avec un coefficient de 0.2. A la figure suivante, un autre exemple montre la coregistration à l'origine et la coregistration obtenue après homogénéisation mise en oeuvre avec un coefficient de 0.2 également. Un œil averti remarquera que les deux images sont mieux jointes et qu'il est maintenant plus difficile de distinguer la jonction des deux images en se basant sur la différence de luminosité du fond de ces deux dernières.

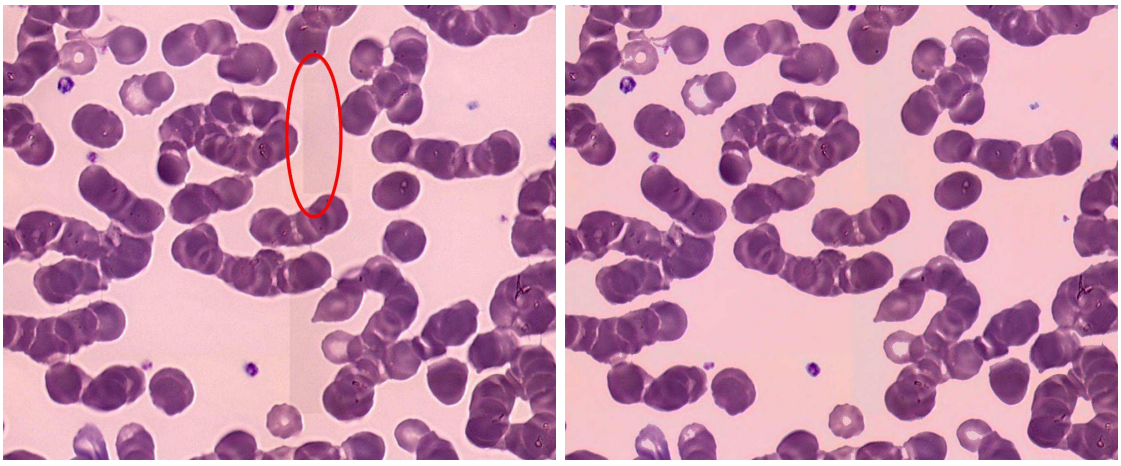

FIG. 38 – Exemple d'homogénéisation

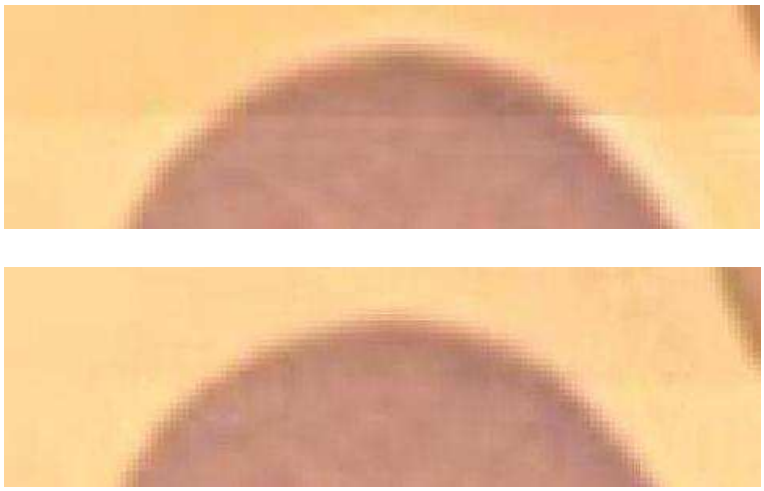

FIG. 39 – Exemple d'amélioration de coregistration

De tels résultats n'ont pas été obtenus dès le premier essai. Comme il a été signalé, diverses techniques ont été utilisées sans apporter de résultats satisfaisants. De plus, et c'est là que la difficulté apparaît, nous avons mentionné dans la partie "Matériel et méthode", qu'un pixel avec une couleur claire possède une valeur de luminosité différente de celle d'un pixel de couleur foncée. En effet, du jaune citron dégagera plus de luminosité que du bleu marine. Or, dans nos méga-images, il y a le fond et les globules. Et chacune de ces classes possède une luminosité différente. Le fond possède une valeur de luminosité plus élevée que celle des globules. Par exemple, dans notre illustration d'amélioration de coregistration de la figure précédente (la figure **??**), un pixel pris au hasard parmi le fond possède une valeur de luminosité de 0.80196 et un autre pixel pris au hasard dans le globule nous donne une valeur de luminosité égale à 0.63333. Il paraît clair que tenter de se rapprocher d'une valeur de référence pour la luminosité de tous les pixels de l'image sans aucune distinction de classe, n'améliorera pas l'image puisqu'elle fera disparaître la différence de luminosité qui existe entre les diverses classes de l'image. Le résultat est une image fade. Dès lors, il faut pouvoir faire la distinction entre les classes de l'image. On en revient aux techniques de seuillage déjà utilisées dans la composition automatique de galeries. Le compromis choisi a été d'homogénéiser uniquement le fond de l'image. Une fois le seuil, permettant de délimiter le fond du reste de l'image, obtenu, la valeur de référence a été calculée par rapport à tous les pixels du fond. Ce sont ces mêmes pixels qui ont vu leur valeur modifiée pour tenter de s'approcher de la valeur de référence obtenue. Un autre problème survient alors, c'est que les globules sont parfois plus clairs en leur centre, et ces pixels plus clairs peuvent être pris pour des pixels appartenant au fond de l'image. Ils seront donc homogénéisés aussi, ce qui a pour effet de supprimer le dégradé qui existe entre les diverses couleurs d'un globule. Un exemple illustre ce problème à la figure 40 où l'image de gauche représente l'image originale et celle de droite l'image homogénéisée.

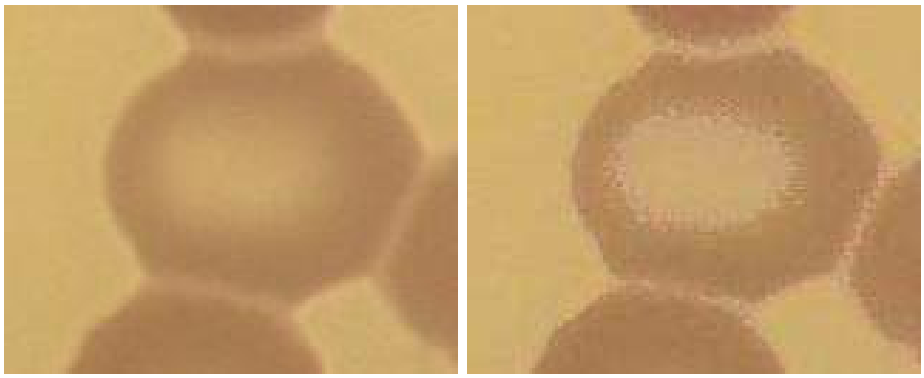

FIG. 40 – Exemple du problème de l'homogénéisation du centre des globules

Afin d'éviter au mieux ce problème, nous allons d'abord extraire les globules de l'image originale. Ensuite l'image est homogénéisée, amenant le problème expliqué ci-avant. Les globules extraits de l'image originale sont ainsi replacés sur l'image homogénéisée pour former le résultat final. Voici l'illustration à la figure 41 de ce que peut donner une homogénéisation sans prise en compte des diverses classes présentes dans l'image. La figure suivante (figure 42) montre l'image des globules extraits et l'image résultat obtenue après homogénéisation en tenant compte des classes.

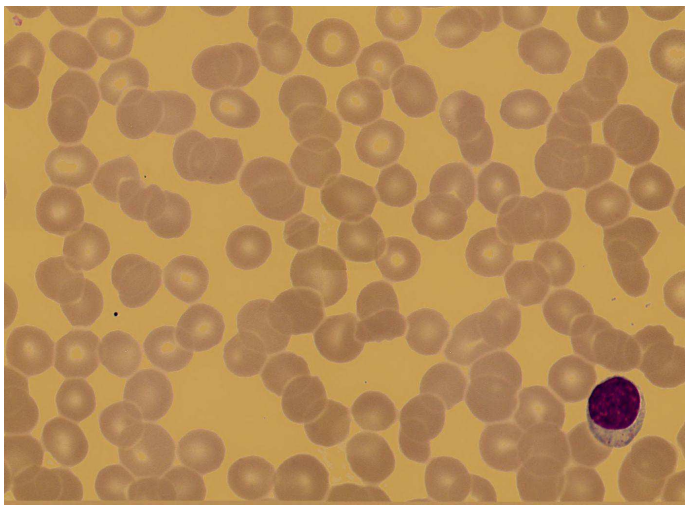

FIG. 41 – Exemple du problème de l'homogénéisation sans prendre en compte les classes de l'image

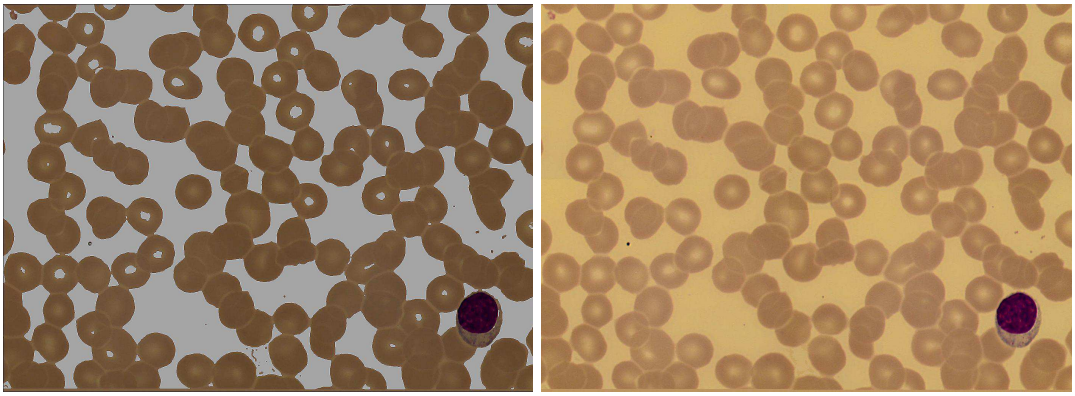

FIG. 42 – Exemple d'extraction des globules et d'homogénéisation prenant en compte ces globules

Au niveau de l'interface homme-machine, le client de visualisation inclut le coefficient d'homogénéisation dans la fenêtre d'ouverture des images (voir annexe D.4).

Voici donc pour l'homogénéisation. Encore une fois les techniques présentées dans la partie "Matériel et méthode" ont bien servi, notamment le codage HSL permettant d'obtenir la luminosité d'un pixel. Et surtout le seuillage permettant de différencier les diverses classes de l'image.

3.5 L'extraction d'images classifiées

En cours de stage, les laborantins ont exprimé le souhait de pouvoir récupérer les images créées par un de leurs logiciels professionnels. Ces images étaient stockées dans une base de données au format .mdb, un format de Microsoft Office Access. Ces images, codées en JPEG, sont stockées dans un champ de type objet OLE (Object Linking and Embedding) et il était impossible de les visualiser, Access nous renvoyant un message d'erreur.

L'objectif premier était simplement de créer un programme pour extraire ces images de la base de données. Puis, petit à petit, d'autres fonctionnalités ont été demandées telles que l'affichage, la classification et l'envoi via DICOM de ces images.

3.5.1 *Extraction des images*

Premièrement, il faut pouvoir extraire les images de la base de données. Encore une fois, le langage utilisé est Java. Et en matière de bases de données, Java est souvent associé à JDBC[27] (Java DataBase Connectivity). Access étant un logiciel de Microsoft, le format permettant la communication d'une application exécutée sous Windows et une telle base de données est ODBC (Open DataBase Connectivity).

[27] JDBC est une API (Application Program Interface) indépendante du SGBD (Système de Gestion de Base de Données) permettant de créer des applications qui interagissent avec des bases de données.

L'application Java créée utilise donc JDBC et ODBC pour communiquer avec la base de données qui nous intéresse. Un gestionnaire ODBC est présent dans les outils d'administration de Windows. Dans cet outil, on peut définir des sources de données en précisant la base de données concernée, le pilote utilisé pour exploiter cette base de données (dans notre cas, Microsoft Access Driver) et un nom pour nommer cette source. La source ainsi définie peut être exploitée dans une application Java par exemple. Le problème est donc le manque de portabilité de cette méthode. En effet, dès que l'application créée est déplacée sur une autre machine, il faut y définir la source de données dans son gestionnaire ODBC. Il existe une astuce permettant de ne pas passer par le gestionnaire ODBC tout en utilisant ODBC. Cette astuce consiste à préciser, dans l'adresse de la base de données, le pilote du SGBD utilisé comme suit :

```
String url= "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};
    DBQ="+cheminDeLaBD;
String driver = "sun.jdbc.odbc.JdbcOdbcDriver";

/*
A la place de ceci :
url = "jdbc:odbc:NomDeLaSourceDansLeGestionnaireODBC";
driver = "sun.jdbc.odbc.JdbcOdbcDriver";
*/
```

Une fois l'application en liaison avec la base de données, il faut exécuter une requête pour obtenir ce que l'on souhaite, dans notre cas les images. Cette requête fournira un *ResultSet*. A partir de ce dernier, il faut appeler la fonction *getBinaryStream(NomDuChamp)* pour obtenir un *InputStream* qu'il restera à transformer en image. Cette image pourra être manipulée ou encore enregistrée. Voici ce que cela donne en Java :

```
InputStream stream = null;
BufferedImage image = null;
/* obtenir le flux binaire du champ jpg de la table images */
stream = rs.getBinaryStream("jpg");
/* convertir ce flux en BufferedImage */
image = ImageIO.read(stream);
```

3.5.2 *Affichage et classification des images*

Dans un premier temps, chaque image extraite de la base de données était enregistrée sur le disque. Après, l'objectif était de pouvoir afficher les images extraites en une galerie d'images, à la manière de la galerie créée précédemment pour les globules blancs et décrite au point 3.3. Ce qui avait alors été développé pour la galerie des globules blancs à été réutilisé et adapté comme nous le verrons dans le point suivant (le point 3.5.3).

Ici aussi, les images contenues dans la base de données sont des images de globules blancs. La galerie composée est donc similaire à la galerie décrite précédemment. Cependant, la base de données apporte des informations supplémentaires telles que la classe de globules dont il s'agit, ou encore les informations du patient et de l'examen correspondant. A partir de ces informations, il était donc possible de classifier les globules contenus dans les images. Des informations contextuelles ont donc pu être ajoutées à la galerie d'images, décrivant la classe à laquelle appartient le globule représenté dans l'image. A partir de ces informations contextuelles, il est devenu possible de faire le lien avec l'outil de comptage des divers globules développé par C. Jadoul et illustré à la figure 7. Connaissant donc le lien entre les diverses lignées de cet outil et les images des globules, un remplissage de l'outil de comptage est possible.

3.5.3 *L'IHM*

L'interface graphique créée consistait, dans un premier temps, en une interface permettant uniquement de sélectionner la base de données contenant les images à extraire et de sélectionner le répertoire où les images seront extraites. Dans un second temps, l'interface d'affichage d'une galerie développée pour le microscope a été ajoutée dans le bas de l'interface graphique. Troisièmement, l'outil de comptage de C. Jadoul a été intégré et se rempli de manière automatique. La figure 43 illustre cette interface où l'on retrouve le champ de sélection de la base de données, le champ de sélection du répertoire d'extraction, une barre de progression et la galerie. Le premier bouton permet de lancer l'outil de comptage en le remplissant automatiquement au préalable. Les deux derniers boutons ont la même fonction que pour le microscope, ils permettent de supprimer ou d'enregistrer une sélection d'images.

Etant donné que la base de données possède diverses informations intéressantes, ces dernières sont automatiquement reprises pour être affichées dans les champs se situant au dessus de la galerie[28]. Ces informations peuvent être modifiées à la guise de l'utilisateur. Les images affichées dans la galerie portent une dénomination (correspondant à la classe du globule), si on double clique sur une image, cette dernière s'affiche en pleine résolution ainsi que sa dénomination. Cette dernière peut être modifiée, il suffit de choisir dans la liste déroulante la dénomination voulue. En cas d'hésitation, plusieurs dénominations peuvent être choisies, auxquelles on peut appliquer un coefficient de certitude. La figure 44 illustre cet affichage en pleine résolution.

[28] dans l'exemple de la figure 43, ces informations n'étaient pas fournies.

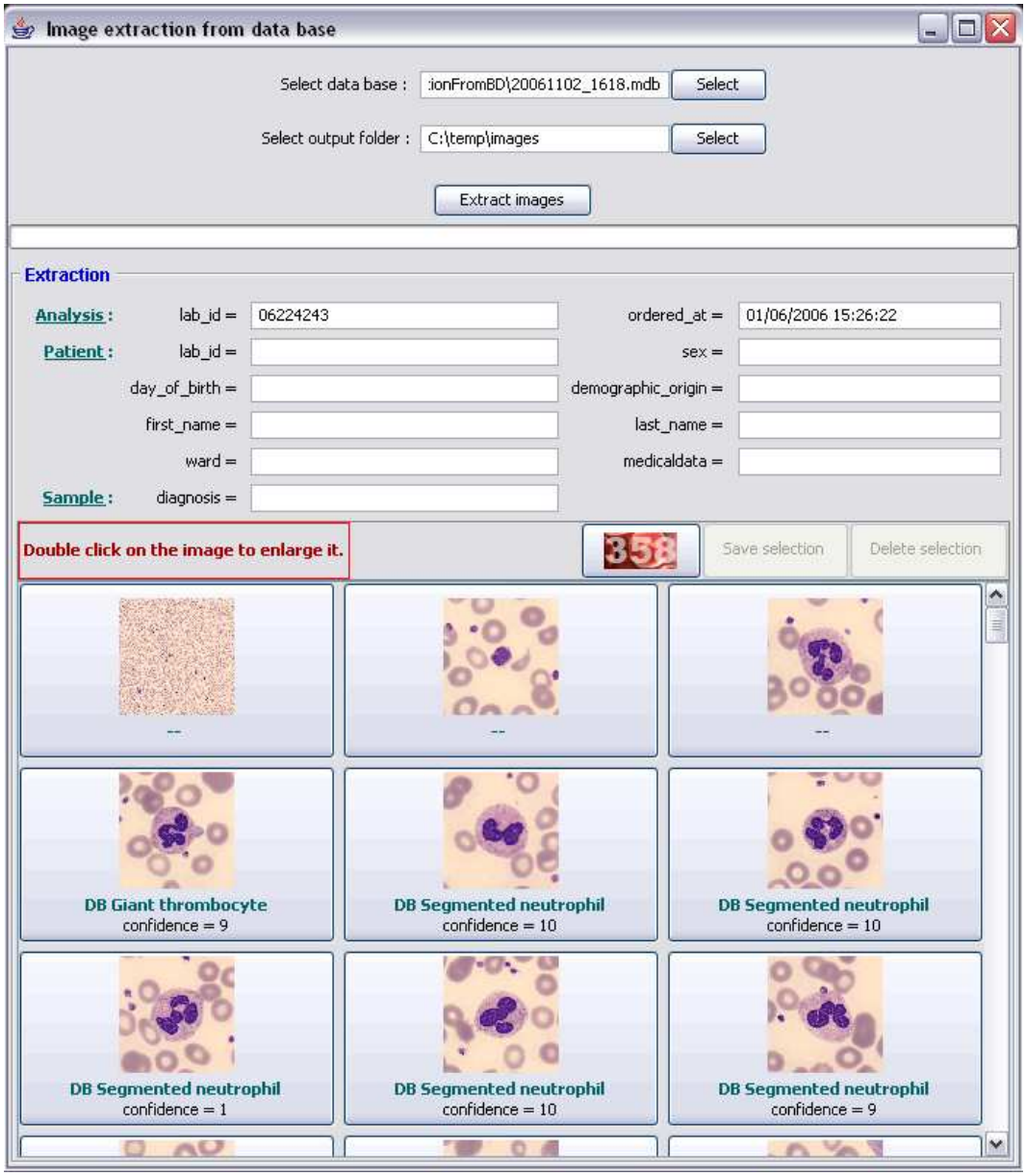

FIG. 43 – IHM de l'extraction d'images

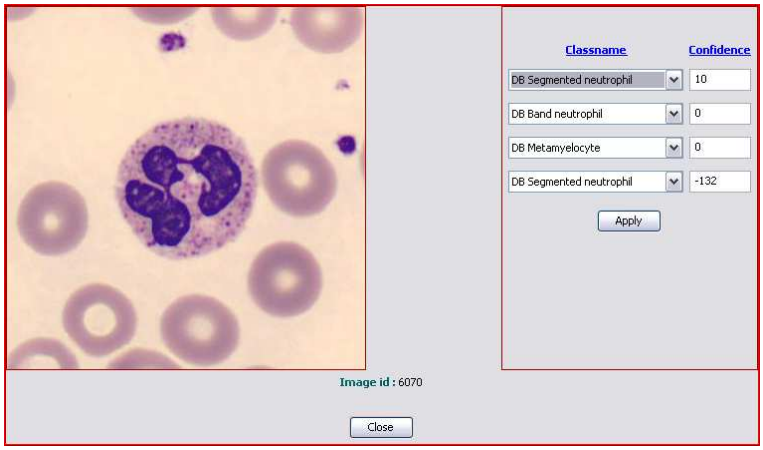

FIG. 44 – IHM de la visualisation d'un globule

3.5.4 *Envoi DICOM*

Un autre objectif demandé concernant l'extraction de ces images est de pouvoir envoyer ces dernières sur le réseau de l'hôpital. Ceci devait pouvoir se faire en utilisant le standard DICOM et devait être développé sous forme de service. Repartant de l'application déjà développée, il suffit d'y ajouter un paramètre au démarrage. Ce paramètre est le chemin vers un répertoire du disque. Ainsi, si le programme détecte un tel paramètre, il ne démarre pas l'interface graphique pour l'utilisateur mais se lance en mode "service", c'est-à-dire qu'il observe le répertoire reçu en paramètre. Dès qu'il remarque qu'une base de données .mdb a été ajoutée, il en extrait les images pour les envoyer sur le réseau. La base de données ainsi traitée est placée dans un répertoire "Terminated" si l'extraction s'est terminée correctement ou dans un répertoire "Failed" si l'extraction a échoué. Ce service observe le répertoire, à la recherche d'une base de données, toutes les 30 secondes et ceci afin de ne pas surcharger inutilement le processeur avec une boucle infinie qui vérifierait sans temps de pause. Les images envoyées sont accompagnées d'informations présentes dans la base de données.

Une fois ces images envoyées sur le réseau à l'aide du standard DICOM, il est possible de les visualiser avec leurs informations contextuelles en utilisant le logiciel Telemis. Ceci peut être observé à la figure 45.

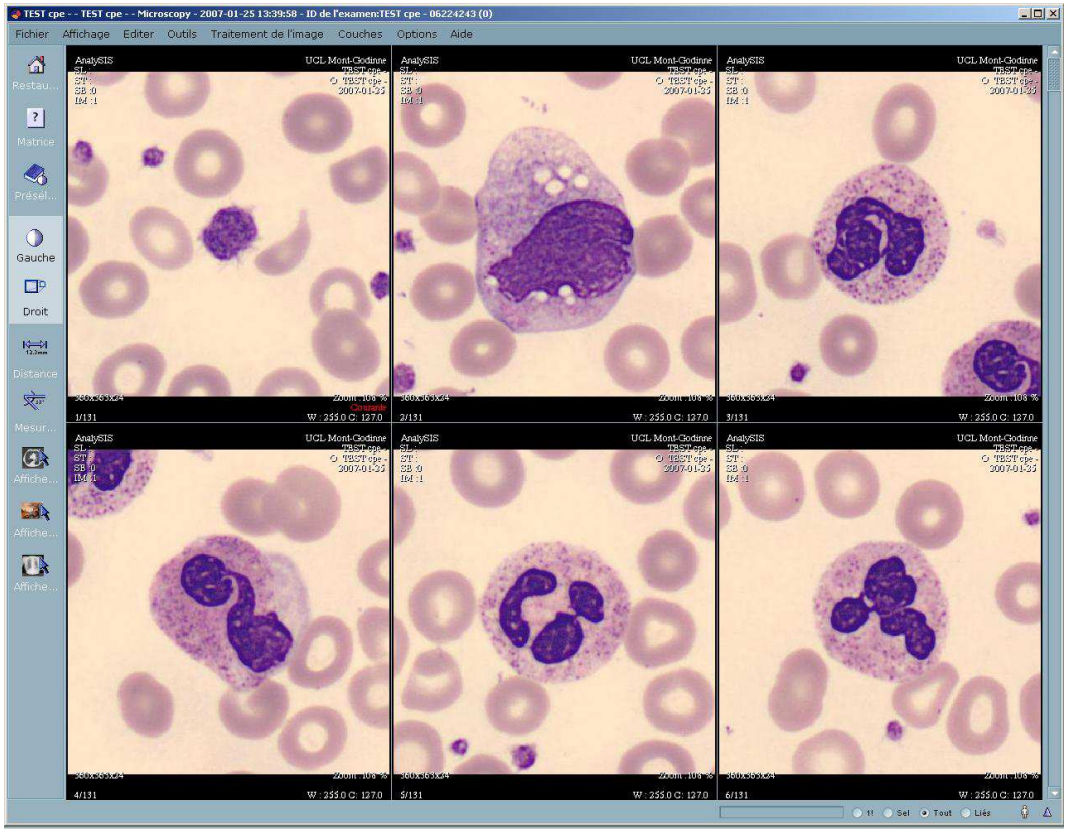

FIG. 45 – Visualisation avec Telemis

3.6 La classification des globules blancs

Notre premier objectif consistait en la création d'une galerie d'images illustrant les globules blancs présents dans un frotti sanguin numérisé. A la croisée des chemins entre cet objectif et celui d'extraction des images classifiées se trouve le souhait de classifier les images de la galerie créée. Tout comme chaque image de la galerie, composée à partir de l'extraction de la base de données, porte une dénomination représentant sa classe, il est intéressant de pouvoir dénommer les images de la galerie construite à partir du microscope. Pour ce faire, il faut connaître la classe du globule illustré dans chaque image. Mener cet objectif à bien permettrait encore une fois de remplir au préalable l'outil de comptage présent dans l'interface du microscope virtuel.

Par manque de temps, cet objectif n'a pas été mené à terme. Néanmoins des pistes ont été explorées. Dans un premier temps, les globules étaient classés dans la galerie en fonction de la taille de la chaîne qui en résulte. C'est d'ailleurs toujours le cas de l'affichage actuel, ainsi les globules sont classés du plus grand au plus petit selon la longueur de la chaîne obtenue lors de l'étape de chaînage.

Comme il a été signalé, une chaîne peut contenir diverses informations. C'est ainsi que la surface du globule (ou plus précisément de son noyau) en nombre de pixels a été ajoutée. Il en va de même pour le taux de remplissage. On entend par taux de remplissage le rapport qui existe entre la surface du carré (ou plutôt devrait-on dire rectangle) dans lequel le globule est inscrit et la surface du globule (toujours en pixels). Cette information peut être exploitée pour avoir une certaine idée de la forme. En effet, un globule bien rond aura un taux de remplissage élevé puisqu'il recouvrira presque toute la surface du carré/rectangle dans lequel il est inscrit. A l'inverse, un globule allongé qui serait placé en diagonale dans le carré/rectangle aura un taux de remplissage faible. Une autre idée d'information est de connaître le rapport qui existe entre la largeur et la hauteur du rectangle en question. Cette information pourra aussi être utile pour déterminer la forme du globule. Effectivement, si le rapport est égal à 1, c'est que la largeur et la hauteur sont les mêmes, ce qui signifie que c'est un carré et donc que le globule possède une forme se rapprochant d'un rond. Si la hauteur est plus grande que la largeur, on assistera alors à un globule allongé dans le sens de la hauteur. Inversement, si la largeur est plus grande que la hauteur, le globule sera allongé dans le sens de la largeur.

L'idée exploitée ici a été d'écrire toutes ces informations dans un fichier Excel au format .xls. Ce fichier Excel, pour un frotti donné contient la liste des images des globules qu'on a pu détecter. Ces images sont stockées dans une colonne à côté de la colonne contenant leur nom. Dans une autre colonne se trouve l'information de la taille de la surface du globule. Puis on rencontre la surface du carré/rectangle dans lequel le globule est inscrit. Ensuite se trouve le taux de remplissage. Et enfin le rapport largeur sur hauteur. Un tel fichier peut donc être exploité en faisant des tris sur ces informations. C'est à ce point que nous nous sommes arrêtés pour le stage. Par la suite, l'idée serait que les tris ainsi dégagés seraient basés sur des critères obtenus de façon automatisée, et pourraient peut-être correspondre à des tris que les laborantins connaissent. Notre perspective est ainsi de pouvoir dégager des classes à partir de ces tris.

Pour créer les fichier .xls avec Java, c'est le package *jxl* qui a été utilisé. La création d'un fichier Excel à l'aide de Java et jxl est expliquée à l'annexe H et est suivie par un exemple de fichier résultat.

Voici donc à quel état d'avancement se situe cet objectif. Passons maintenant au dernier objectif cité.

3.7 L'autofocus

Lors de l'acquisition des images permettant de créer une méga-image, le microscope fait appel à l'autofocus hardware afin d'obtenir l'image la plus nette possible. Cependant, cet autofocus ne fonctionne pas toujours de manière idéale. En effet, il est possible que l'autofocus n'arrive pas à déterminer la meilleure image. Ce dernier fonctionne en déterminant une zone de focus de forme rectangulaire au centre de l'image à acquérir, aussi appelée

zone de mise au point. Il compare cette zone pour chaque image qu'il observe en montant et en descendant l'objectif du microscope. Néanmoins, si la zone de focus ne possède aucun élément contrasté en son sein, c'est-à-dire que cette zone est apparentée à un aplat d'une même couleur, alors l'autofocus n'arrive pas à différencier les images. Il lui est donc impossible de déterminer l'image la plus nette et un message le signalant est retourné. La figure 46 illustre ce phénomène. L'image montre ce qui est observable au microscope et illustre la zone de focus. Cette dernière est clairement dans une zone où il sera impossible de comparer la netteté de la zone, quelle que soit la mise au point.

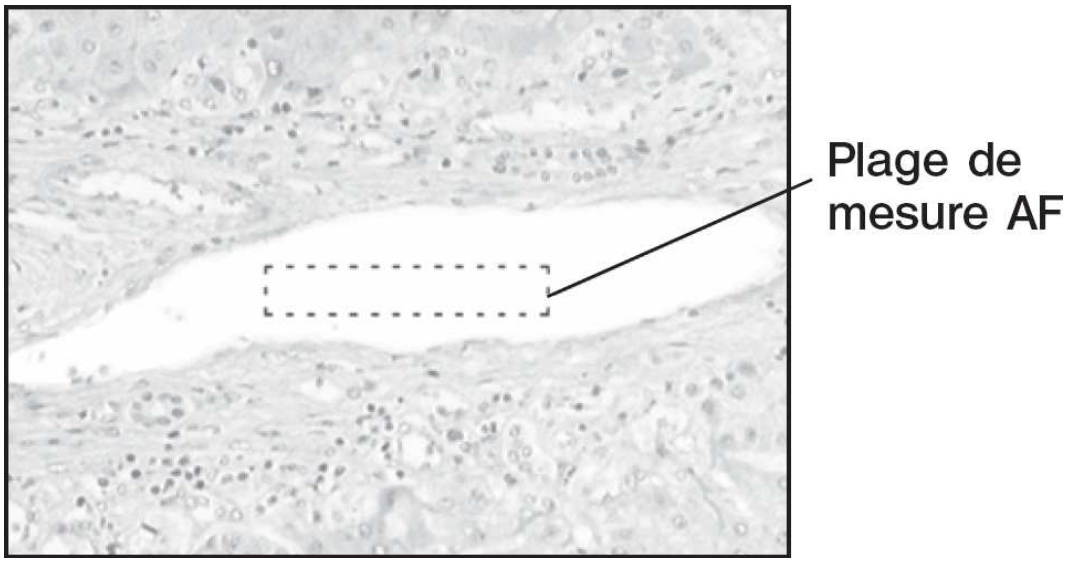

FIG. 46 – Problème de mauvaise mise au point avec l'autofocus

L'objectif ici était donc de pouvoir palier à ce problème en programmant un module AnalySIS à l'aide du langage Imaging-C qui lancerait l'autofocus software du logiciel Analysis. Ainsi, dès que l'autofocus hardware n'obtient aucune image nette, l'autofocus software est lancé. Seulement, l'autofocus software n'est pas assez précis pour l'objectif 100× car il se déplace par pas de $0.2\mu m$ selon l'axe Z, or avec un tel objectif, un déplacement de $0.1\mu m$ est significatif.

Une autre idée était de créer un module d'autofocus software de toute pièce à l'aide des commandes RS232 disponibles pour l'autofocus hardware. Notamment grâce aux commandes "MOV" permettant de se déplacer selon l'axe Z, la commande "AFP ?" permettant de connaître la position de l'autofocus, et la commande "POS" permettant de positionner l'autofocus. Ainsi, en récupérant la position de l'autofocus avant que ce dernier ne s'égare, il était possible de le replacer et d'avancer selon l'axe Z petit à petit en comparant chaque image observée. Pour déterminer l'image la plus nette, l'idée était de compter le nombre de valeurs différentes pour l'ensemble des pixels de l'image observée, en supposant que l'image la plus nette compterait le plus grand nombre de valeurs différentes. Néanmoins, les commandes RS232 décrites ci-dessus ne fonctionnaient pas toutes. En effet, la com-

mande POS et la commande "AFP ?" retournaient un message d'erreur. Après discussion avec des représentants de chez Olympus, il s'est avéré que notre autofocus n'était pas à jour et donc ne possédait pas les fonctionnalités souhaitées. De plus l'espoir mis dans la fonction "AFP ?" était vain car même si nous avions disposé de cette dernière, elle ne nous aurait pas aidé car sa précision ne va pas en deçà de $1\mu m$.

Cet objectif n'a donc pu être réalisé, néanmoins nous avons mis le doigt sur le problème. Pour de plus amples informations concernant le fonctionnement de l'autofocus et des commandes RS232, s'en référer aux manuels d'Olympus [Oly05] et [cL99].

Ceci conclu la partie présentant les résultats obtenus pour chaque objectif décrit dans la partie "Cadre et objectifs".

Quatrième partie

p

Discussion

4 Quatrième partie : discussion

4.1 Introduction

La partie précédente, intitulée "Résultats", nous explique comment mettre en oeuvre les techniques présentées dans la partie "Matériel et méthode" pour mener à bien les objectifs décrits dans la partie "Cadre et objectifs". Nous avons ainsi pu voir une application qui permet de créer une galerie d'images à partir d'une méga-image, une technique pour homogénéiser les images et une application pour extraire les images d'une base de données.

Il est évident que ces résultats sont améliorables. En effet, ils ont été obtenus sur une période de plus ou moins 5 mois, ce qui n'a pas été assez suffisant pour atteindre tous les objectifs. Cette présente partie s'emploie donc à poser un regard critique sur les résultats obtenus et à proposer des idées d'améliorations et des perspectives de développements complémentaires.

4.2 Réalisation des objectifs

L'objectif de départ concernait la réalisation d'une galerie d'images à partir d'une méga-image représentant un frotti sanguin. Cet objectif a pu être mené à terme. Ainsi, le client de visualisation du microscope virtuel inclut maintenant un outil de création d'une galerie à partir du frotti visualisé.

L'homogénéisation d'images est aussi un objectif qu'il a été possible de mener à bien après avoir testé diverses idées.

En ce qui concerne l'objectif demandant d'extraire simplement les images d'une base de données, il a pu être rapidement réalisé. C'est donc dans une certaine continuité que cet objectif a été étoffé afin de permettre la visualisation des images et la récupération d'informations à partir de la base de données. Un autre objectif est venu se greffer demandant de pouvoir créer un service permettant d'envoyer les images extraites à l'aide de DICOM sans passer par l'interface graphique permettant la visualisation des images. Aussi bien l'application de visualisation des images que le service d'envoi des images ont pu être réalisés.

Pour en revenir à la création de galeries d'images, un nouvel objectif était de pouvoir classifier les images présentes dans une galerie. Cet objectif n'a pas été réalisé dans son entièreté, mais des pistes ont été explorées et des idées ont été données.

Par contre, l'objectif de correction de l'autofocus n'a pu être mené à bien malgré le temps passé dessus. Le problème se situant au niveau du firmware embarqué dans le boîtier d'autofocus, nous sommes restés impuissants face à ce problème. Au final, la solution adop-

tée jusqu'alors reste pour l'instant la seule permettant d'acquérir des images plus ou moins nettes. Cette solution consiste à interrompre l'acquisition quand l'autofocus échoue et de demander à l'utilisateur de replacer manuellement l'autofocus. Cette solution implique la présence d'un acteur humain lors de l'acquisition alors que sans ce problème, cet acteur pourrait vaquer à d'autres occupations puisque l'acquisition pourrait se faire de façon entièrement automatique.

A vrai dire, les objectifs réalisés sont améliorables, c'est ce que nous verrons au point suivant (point 4.3). D'autre part, les objectifs n'ayant pas été atteints font l'objet de perspectives d'avenir. Ces perspectives sont décrites au point 4.4.

4.3 Limites et améliorations possibles

4.3.1 *La création de galeries*

A partir de l'interface du client de visualisation du microscope virtuel, il est maintenant possible de créer une galerie d'images. Ces images reprennent les globules blancs du frotti visualisé.

Comme le mentionne son intitulé, le client de visualisation est un client. En effet, il doit pouvoir être utilisé par diverses personnes sur diverses machines. Ceci signifie que les traitements lourds se situent sur le serveur et que c'est le client, en émettant des requêtes, qui déclenche ces traitements. Or dans notre cas, la création de galeries se fait directement chez le client. Le temps pour la création d'une galerie variera donc d'une machine à l'autre. Il serait intéressant de déplacer ce traitement au niveau du serveur. Ce dernier créerait les galeries et en enverrait les images au client qui n'aurait plus qu'à les afficher. De plus, dès que le serveur aurait créé une galerie, il pourrait la sauvegarder pour ensuite la renvoyer lors de futures requêtes et ainsi éviter de recréer une galerie déjà conçue.

Comme on vient de le signaler, il s'agit d'un client, donc il doit pouvoir fonctionner sur un maximum de configurations de machines. Or ces configurations comprennent diverses résolutions d'écran, ce qui n'est pas pris en compte par notre interface graphique. En effet, la galerie d'images a été ajoutée en bas de cette interface, agrandissant ainsi la hauteur de cette dernière. Or il serait judicieux de limiter la taille de l'interface graphique de façon à pouvoir l'afficher sur une majorité de machines utilisant toutes sortes de résolutions. Ainsi, peut-être faudrait-il retirer la galerie de l'interface principale et la placer dans une nouvelle fenêtre qui apparaîtrait uniquement lorsque cette dernière serait invoquée, à l'image de l'outil de comptage de C. Jadoul. Le bouton de création de galerie serait ainsi ajouté à la barre de boutons se situant dans la partie supérieure de l'interface. Cliquer sur ce bouton lancerait la requête de création de galerie au serveur et afficherait la galerie obtenue dans une nouvelle fenêtre. Cette fenêtre pourrait prendre la forme d'une *JWindow*[29] afin de lier cette dernière à l'interface principale et, de ce fait, ne pas se retrouver avec plusieurs fenêtres ouvertes, indépendantes les unes des autres.

[29]Pour lier une JWindow avec l'interface principale, s'en référer à l'annexe I.3.

Un autre problème de la création d'une galerie est le temps nécessaire à cette création. En effet, le traitement d'image sur des images de la taille d'une méga-image nécessite un certain temps pour être effectué. De plus, le nombre de traitements à appliquer les uns à la suite des autres augmente donc le temps requis pour la création d'une galerie. Voici quelques idées d'amélioration pour diminuer le temps de création :

- Tester plus longuement les méthodes de seuillage afin d'en trouver une qui permettrait d'obtenir directement le seuil qui délimite les globules blancs du reste ;
- Ne plus passer par l'étape de binarisation de l'image et donc passer directement au chaînage. Ce chaînage se ferait sur les pixels blancs obtenus après l'étape de détection de contours ;
- Améliorer le chaînage, peut-être en essayant de déterminer la direction du contour afin de diminuer le nombre de comparaisons à faire. En effet, actuellement une comparaison se fait pour chaque pixel adjacent à celui traité afin de savoir si oui ou non le pixel adjacent comparé fait partie du contour. Ce qui veut dire qu'il y a 8 comparaisons pour chaque pixel du chaînage. Peut-être faudrait-il essayer, lorsqu'on rencontre un embranchement, de lancer autant de threads qu'il y a d'embranchements. Chaque thread s'occuperait de chaîner l'embranchement dont il est responsable. Ceci éviterait d'extraire un à un chacun de ces embranchements et ensuite les relier entre-eux. Une autre voie à explorer serait de tester les modèles déformables afin, peut-être, de remplacer le chaînage si cette technique s'avère plus satisfaisante. Cette technique consiste à utiliser un modèle géométrique de manière à ce qu'il suive les contours ;
- Mettre en place un système de mémorisation des tiles déjà téléchargées. En effet, actuellement on récupère auprès du serveur les tiles intéressantes pour la détection des globules. Une fois les globules détectés, les tiles permettant de créer les images de la galerie sont téléchargées. Ceci est une redondance, il serait intéressant de ne télécharger que les tiles qui n'ont pas encore été téléchargées ;
- Java est connu comme étant un langage d'une certaine lourdeur et d'une certaine lenteur[30]. Il faudrait peut-être essayer de programmer les traitements d'image dans un langage plus performant tel que le langage C, voir le langage C++ pour les accros de l'orienté objet. Cependant je dois admettre que l'application actuelle a été programmée dans un but fonctionnel et non de performance. Le code s'en trouve donc plus lisible qu'optimisé[31].

Un autre problème de la création d'une galerie est la détection de bruit. En effet, les globules blancs sont détectés dans la méga-image, mais d'autres éléments ne correspondant pas à de tels globules sont aussi détectés. Nous avons vu qu'un filtre médian est utilisé afin de retirer une partie du bruit de l'image lors de l'étape de filtrage. Il faudrait peut-être tester avec divers filtres médians plus grands, afin d'en déterminer un optimal, au risque de perdre en performance de temps lors de cette étape. Ce temps dépensé serait peut-être regagné par la suite puisqu'il ne faudrait plus appliquer les étapes suivantes sur ce que l'on

[30] Cependant les performances de Java vont en s'améliorant au fur et à mesure des versions de la JVM.

[31] Il faut bien avouer que, de manière générale, la culture des programmeurs en Java (fiabilité, lisibilité et compréhensibilité du code) est loin de la culture des programmeurs en Assembleur (performance et optimisation).

considère comme étant du bruit. Une autre idée serait de ne pas utiliser de filtrage médian et de le remplacer par un opérateur morphologique (voir point 2.3.2 dans la partie "Matériel et méthode") après la binarisation de l'image si l'étape de binarisation est conservée. Cet opérateur serait l'érosion afin de supprimer le bruit. Ici encore il faudrait déterminer la taille de l'élément structurant par divers tests.

La gestion de la mémoire en Java :
Un problème délicat rencontré lors du développement de la création d'une galerie fut le problème de dépassement de la mémoire. Le garbage collector de Java permet au programmeur de ne pas se soucier de la gestion de la mémoire et de ne pas avoir à penser à libérer les ressources utilisées tel qu'il est nécessaire de le faire avec un langage comme le langage C par exemple. Ceci peut être considéré comme un avantage pour le programmeur qui n'a pas à se soucier de cet aspect, cependant le reproche que l'on peut faire est que le programmeur n'a qu'un contrôle limité de la gestion de la mémoire.

Dans le cas de la création d'une galerie, les images la composant sont obtenues à partir d'une méga-image. Autrement dit, une méga-image, qui par définition possède une grande taille, peut produire une galerie comportant un nombre important d'images. Comme il a été expliqué, la création d'une galerie fait appel aux tiles intéressantes. Il y a donc plusieurs tiles à traiter les unes après les autres. Si on ne prête aucune attention à la gestion de la mémoire, celle-ci croit de plus en plus jusqu'à produire une *java.lang.OutOfMemoryError*. Une telle exception est levée lorsque la JVM ne peut plus allouer de mémoire et que le garbage collector ne peut plus en libérer.

Une solution possible et largement utilisée est de modifier la taille maximale de la mémoire au démarrage de la JVM à l'aide de l'option "-Xmx" et en lui précisant la taille maximale à utiliser. Néanmoins, ceci contourne le problème car si une telle erreur survient, il est probablement nécessaire de vérifier le code de l'application afin de détecter et de corriger quelques mauvaises pratiques de programmation causant cette erreur[32].

Pour éviter au mieux les fuites mémoire, il faut affecter la valeur *null* aux objets qui ne sont plus utilisés. Utiliser la méthode *flush()* sur une *BufferedImage* ne libérera la mémoire octroyée que lors du passage du garbage collector, il vaut donc mieux forcer le garbage collector à passer à l'aide de *gc()*.

Par défaut, Java utilise des *hard references* (aussi appelées *strong references*). Ces références existent tant qu'elles ne sont pas placées à *null* ou tant que le cheminement du programme ne les fait pas disparaître (par exemple lors de la fin de la méthode dans laquelle cette référence est déclarée). Un objet n'est supprimé de la mémoire que si toutes ses *hard references* sont supprimées. Il existe cependant les classes *java.lang.ref.WeakReference*,

[32]Une idée pour détecter une fuite mémoire est d'afficher la mémoire utilisée par l'application en cours d'exécution à divers endroits stratégiques. Si la différence de mémoire utilisée est flagrante, c'est qu'il y a peut-être un problème qui survient entre ces deux affichages. Pour afficher la mémoire en Java, s'en référer à l'annexe I.4.

java.lang.ref.SoftReference et *java.lang.ref.PhantomReference*. Les *WeakReference* autorisent la libération d'un objet si toutes les références à ce dernier sont de telles références. Les *SoftReference* permettent au garbage collector de libérer l'objet référencé si cela est nécessaire (par exemple avant une *OutOfMemoryError*). Pour leur part, les *PhantomReference* permettent de savoir si un objet a été ou non libéré de la mémoire. Il serait donc prudent de repenser l'application de création d'une galerie en utilisant ces types de références.

Une autre source de consommation de la mémoire sont les interfaces graphiques. Il faut toujours bien faire attention à utiliser la méthode *dispose()* lorsqu'on quitte une fenêtre afin de libérer la mémoire utilisée par cette dernière. Etudier diverses API d'interfaces graphiques pourrait être utile afin d'en dégager celle qui est la moins gourmande en mémoire[33].

4.3.2 *L'homogénéisation d'images*

Cet objectif consistait en l'amélioration des méga-images composées à partir des images capturées au microscope. Il s'agissait d'homogénéiser la luminosité de ces images afin d'atténuer la jonction toujours visible entre les images composantes. Cet objectif a pu être mené à bien, cependant il est possible d'y apporter quelques améliorations. En effet, comme il a été expliqué, cette application commence par extraire les globules de la méga-image afin d'homogénéiser le fond de cette méga-image. Ensuite les globules sont replacés sur ce fond homogénéisé. Cette séparation des globules et du fond trouve son explication dans le fait que ces derniers ont une luminosité différente de celle du fond.

Les méga-images obtenues suite à l'homogénéisation atténuent la jonction des images composantes si l'on observe le fond. Cependant, étant donné que les globules ont été extraits au préalable, la jonction des images est toujours observable à l'intérieur même des globules. Ceci peut être observé à la figure 47 où l'image d'origine montre clairement les jonctions et où l'image homogénéisée montre les jonctions dans les globules.

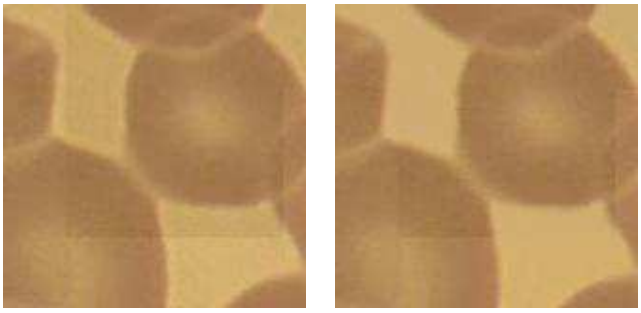

FIG. 47 – Exemple de jonction visible dans les globules

Une voie à explorer serait de calculer plusieurs valeurs de luminosité de référence.

[33] A ce titre, il serait intéressant d'évaluer les possibilités de SWT (Standard Widget Toolkit). Cette API me semble offrir une plus faible consommation mémoire

Chaque valeur correspondrait à une classe différente de la méga-image (typiquement, le fond, les globules blancs et les globules rouges). Néanmoins, la difficulté de cette approche résiderait probablement dans le fait que les globules possèdent, en leur sein même, des couleurs dégradées.

Pour obtenir la luminosité d'un pixel, il faut passer par le calcul des trois composantes HSL. Il y a peut-être moyen de gagner du temps en trouvant une formule permettant d'obtenir directement la luminosité sans passer par le calcul des trois composantes.

4.3.3 *L'extraction d'images d'une base de données*

Il était question, ici, d'extraire des images enregistrées dans une base de données afin de pouvoir les visualiser ou de les envoyer sur le réseau de l'hôpital en utilisant DICOM.

L'application créée va rechercher les champs nécessaires pour obtenir les images et les informations contextuelles. La limitation posée par cette application trouve sa traduction dans le fait que la base de données doit toujours respecter la même structure, auquel cas il lui serait impossible d'extraire les images de cette dernière. Normalement, dans le cas d'une base de données à la structure non-conforme, une boîte de dialogue survient lors de l'extraction signalant que cette dernière a échoué. Cependant ce message n'est pas assez précis et il serait judicieux d'affiner le niveau de détection des erreurs. Ceci permettrait de pouvoir signaler à l'utilisateur si l'erreur provient de la base de données ou de l'application même.

Si jamais la structure de la base de données devait changer à l'avenir, l'idéal serait que l'application fasse fi de ces changements. Ceci éviterait de devoir réadapter l'application à la nouvelle structure de la base de données. S'il est possible de récupérer les méta-données de la base de données, il est donc possible de parser ces dernières afin de rechercher l'emplacement des données recherchées et d'adapter les requêtes d'extraction en conséquence.

Une autre limitation rencontrée par cette application réside dans le fait qu'il est possible de fournir des informations au standard DICOM afin de contextualiser les images envoyées. Actuellement, seules quelques informations sont fournies en les récupérant de la base de données. Il est certainement possible de récupérer d'autres informations intéressantes à utiliser pour décrire le contexte des images. Il faudrait donc rechercher ces informations dans la base de données ou encore les demander aux laborantins si ces informations ne sont pas présentes dans cette base. Une description des champs utilisés se trouve à l'annexe D.5.

Ceci conclut la mise en évidence des limites et la présentation d'idées pour repousser ces limites.

4.4 Perspectives

Nous venons de parcourir les limites observées des diverses applications ainsi que des idées à explorer pour éviter ces limites. Ci-suit, diverses perspectives sont décrites afin de compléter ce qui a été développé.

4.4.1 *La classification des globules blancs*

Nous avons une application permettant de créer une galerie d'images des globules blancs. Comme nous le savons, ces globules blancs sont de diverses sortes : les lymphocytes, les basophiles, les eosinophiles, les neutrophiles et les monocytes. La perspective la plus intéressante dans la continuité de ce travail serait de pouvoir créer une galerie d'images qui pourrait classifier ses images de globules blancs selon les diverses sortes.

L'idée que nous avions abordée dans le point 3.6 consistait a effectuer divers classements dans des fichiers Excel sur base de critères obtenus lors du chaînage pour peut-être en dégager des classements significatifs.

D'autres critères de classement à prendre en compte serait la taille des globules. Ainsi, si les images acquises le sont toutes à l'aide de l'objectif 100×, alors on sait calculer la taille de ces globules étant donné qu'on connaît la taille d'un pixel. Il faudrait donc pouvoir déterminer, pour un globule, combien de pixels de large composent ce dernier. Bien entendu, il faut faire la supposition que l'acquisition des images se fait toujours avec l'objectif de grossissement 100×. Cependant, on pourrait imaginer une information contextuelle contenue dans des méta-données qui permettrait de connaître l'objectif de grossissement utilisé et ainsi d'adapter la mesure de la taille grâce à cette information.

De plus, si on possède l'information contextuelle du grossissement, il est maintenant possible de fixer un seuil sous lequel les chaînes obtenues lors du chaînage seraient supprimées. Effectivement, jusqu'alors, cette taille limite n'était pas déterminée et donc les chaînes que l'on pourrait nommer "chaîne bruit" n'étaient pas supprimées. Supposons qu'une chaîne de longueur de 10 pixels pourrait être considérée comme étant du bruit, et donc comme étant à supprimer, avec un objectif 100×. Une chaîne de cette taille ne serait peut-être pas du bruit si l'objectif utilisé pour l'acquisition est un objectif 40×. Si l'information contextuelle, permettant de connaître l'objectif utilisé, pouvait être récupérée, il serait donc possible d'adapter le seuil d'élimination d'une chaîne bruit selon cette information.

Une autre voie à explorer serait celle de la détermination de la texture des globules. La texture se réfère à l'arrangement et à la fréquence de la variation de teinte. Ainsi serait-il peut-être possible de classifier certains globules via ce critère. La figure 48 nous montre deux globules à la texture différente.

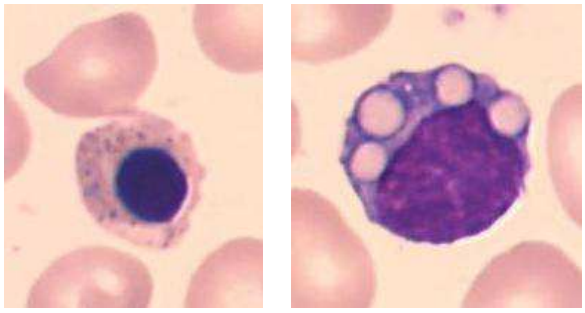

FIG. 48 – Exemple de textures différentes

Une fois les images de la galerie classifiées, on se rapprocherait fortement de la galerie composée dans l'application d'extraction d'images à partir d'une base de données. Peut-être pourrait-on alors faire un rapprochement entre ces deux applications où seule la source de données changerait. une première source serait les méga-images, une seconde source serait les bases de données. Ainsi, la fonctionnalité d'envoi des images à l'aide de DICOM pourrait être modifié et adapté pour les images sélectionnées d'une galerie. Dans la continuité, si les images sont triées selon leur classe, il est possible de remplir l'outil de comptage, ce qui consiste une source d'informations enrichissante. Ces informations de comptage pourraient aussi être envoyées à l'aide de DICOM, peut-être au format XML (eXtensible Markup Language).

4.4.2 *Le protocole JPIP*

JPIP (JPEG2000 Interactive Protocol) est un protocole de communication qui permet le transfert d'images JPEG2000 complètes ou partielles. Ce protocole est décrit dans la partie 9 du standard JPEG2000.

Jusqu'alors, le protocole utilisé entre le client de visualisation et le serveur de visualisation est TCP (Transfert Control Protocol). Etant donné que les méga-images utilisées sont codées en JPEG2000, il faudrait étudier l'éventualité d'utiliser JPIP pour remplacer TCP comme protocole de communication entre ces entités.

4.4.3 *L'homogénéisation dès l'acquisition*

Avec le client de visualisation, il est possible de visualiser les méga-images homogénéisées. Pour ce faire, le client de visualisation demande au serveur de visualisation une méga-image homogénéisée qui, à son tour, demande au serveur de création de recréer la méga-image avec le coefficient d'homogénéisation précisé par le client.

Une perspective intéressante serait d'introduire le coefficient d'homogénéisation dès l'étape d'acquisition de l'image. De cette façon, le serveur de création pourrait créer une image homogénéisée dès le départ à l'aide du coefficient donné.

Remarque : tant qu'il est question d'acquisition de méga-images à partir du microscope, il se peut que les méga-images obtenues incluent une mauvaise coregistration comme illustré en annexe (voir la figure52 à l'annexe C). Bien qu'il soit possible de corriger l'application du serveur de création de façon à ce qu'il assemble les images composantes de façon idéale, ce problème trouve sa source ailleurs. En effet, le problème ne vient pas du serveur de création mais bien de l'acquisition des images composantes par le microscope. Si ce problème survient il convient de recalibrer la caméra du microscope.

4.4.4 *L'autofocus*

Le problème de l'autofocus qui échoue lorsque la zone de mise au point n'est pas idéale n'a pas pu être résolu car le firmware de l'autofocus ne permettait pas d'utiliser certaines commandes RS232.

L'idée a donc été de pouvoir lancer l'autofocus software dès que l'autofocus hardware échoue. Cependant cette idée n'a pu aboutir car l'autofocus software ne fonctionne pas lorsque l'objectif de grossissement est l'objectif 100×. Or cet objectif est justement celui utilisé pour l'acquisition des images.

Une perspective d'avenir serait d'essayer de développer son propre autofocus software qui pourrait peut-être être plus précis que celui existant. L'idée serait, lorsque l'autofocus atteint la limite de l'axe Z et que l'erreur est retournée, de déplacer la platine selon l'axe Z. Ainsi, en bougeant la platine et en capturant des images lors de ce déplacement, une recherche dichotomique pourrait être effectuée sur base de ces images jusqu'à trouver le focus permettant d'obtenir une image nette. Le problème est de voir s'il est possible de déplacer les platines selon l'axe Z de façon suffisamment précise à l'aide de la commande "MOV". Etant donné qu'il n'est actuellement pas possible de connaître la position exacte de l'autofocus, s'il est possible de déterminer les limites de ce dernier, sa position sera connue une fois qu'il aura atteint une de ces limites. A partir de là, lancer la recherche dichotomique permettra aussi de connaître sa position par rapport à la limite atteinte. Un autre problème est donc de voir s'il est possible de connaître les limites de l'autofocus.

Conclusion

c

Conclusion

La télémédecine est une discipline en constante évolution. L'amélioration continue de cette discipline conduit à une synergie d'entités distantes diverses dont le bénéficiaire final est le patient. Dans cette lignée, le programme de recherche en télémicroscopie des cliniques universitaires UCL Mont-Godinne vise l'automatisation des analyses cyto-hématologiques.

L'informatique trouve, de toute évidence, sa place dans ce domaine. Les travaux de recherche précédents utilisaient la communication avec le firmware du microscope via commande RS232, les communications réseau entre les diverses entités du système, des algorithmes de coregistration pour la composition de méga-images, l'utilisation d'un système de cache pour le chargement d'images ou encore l'utilisation du standard JPEG2000 pour l'encodage des images. Toutes ces techniques ont mené à une application de microscopie virtuelle mettant en oeuvre la télémicroscopie de type "store and forward".

Dans la continuité de ces travaux, effectués dans le laboratoire d'hématologie de Mont-Godinne, mon stage consistait à recréer une galerie d'images à partir d'une méga-image. Les images de cette galerie doivent représenter les globules-blancs contenus dans cette méga-image afin d'aider les analyses cyto-hématologiques.

L'informatique apporte son soutien à la création de telles galeries à l'aide des traitements d'image. C'est dans ce cadre que ce mémoire a abordé divers traitements d'image tels que le filtrage, le seuillage, la détection de contours ou encore le chaînage de contours et qu'il a décrit comment utiliser ces traitements afin d'obtenir une galerie des globules blancs.

Nous avons commencé par décrire le cadre de travail dans lequel mon stage a été effectué. Suite à cela, les objectifs du stage ont été énoncés. La partie suivante nous a décrit le matériel et les méthodes utilisées en se concentrant principalement sur le traitement d'image. Puis, nous avons vu comment mettre en oeuvre les techniques décrites pour atteindre les objectifs fixés. Et finalement, nous avons porté un regard critique et constructif sur ce qui a été réalisé. C'est ainsi que diverses idées d'améliorations et d'ajouts ont pu être données.

Bien qu'au final l'application créée peut produire une galerie d'images des globules blancs, nous ne nous sommes pas arrêtés là. Effectivement, les traitements d'image ayant été étudiés, nous avons ensuite vu comment améliorer, à l'aide de certains traitements, les méga-images obtenues par coregistration. Au final, ce mémoire décrit comment obtenir une galerie d'images à partir d'une méga-image mais aussi comment améliorer la qualité de ces méga-images.

De plus, nous avons vu comment extraire des galeries d'images déjà existantes mais contenues dans des bases de données. Cette partie du travail décrit une application ca-

pable de transmettre les images à l'aide de DICOM.

Toujours dans le cadre de la télémédecine, les cliniques universitaires de Mont-Godinne collaborent avec Telemis S.A. dont l'activité se situe entre l'archivage d'images, la communication d'images et la télémédecine. Nous avons abordé, dans la partie d'extraction d'images d'une base de données, la visualisation de ces images avec le logiciel Telemis après transmission via DICOM. Ceci nous a amené à la perspective intéressante de la classification et de la communication des images présentes dans une galerie créée à l'aide de notre application.

D'un point de vue plus personnel, ce stage fût passionnant car il m'a permis de découvrir un domaine, celui de la médecine, qui m'était jusqu'alors inconnu et aussi d'étudier et d'utiliser les traitements d'image. De plus j'ai appris à faire des recherches, dans cette discipline qui m'était inconnue, afin de trouver des solutions aux problèmes posés.

L'écriture de ce mémoire fût aussi une expérience intéressante car ce n'est pas toujours évident d'expliquer, de manière abordable, les résultats obtenus grâce à la recherche.

Pour terminer, nous pouvons dire que le laboratoire d'hématologie de Mont-Godinne est un laboratoire performant et notamment dans le contrôle de qualité. Le microscope virtuel qui est mis en place, à l'élaboration duquel j'ai eu la chance de participer, est une application prometteuse qu'il ne faut pas négliger.

Annexes

a

Annexes

A Mega-image

Une méga-image est une image de taille conséquente. Un frotti sanguin numérisé est composé de plusieurs images acquises par le microscope. Ces dernières sont coregistrées afin de former une méga-image. Cette annexe présente un exemple de méga-image.

Premièrement, la figure 49 est centrée sur un globule blanc et illustre la taille de ce dernier en résolution maximale obtenue à l'aide de l'objectif 100×. On aperçoit des globules rouges entourant le globule blanc et aussi une plaquette (petite "tache violacée" en bas).

Deuxièmement, la figure 50 montre la méga-image entière à partir de laquelle la figure précédente à été tirée. Cette méga-image est composée de 20 images, capturées par le microscope, en largeur et 20 images en hauteur. Elle est donc composée d'un total de 400 images capturées par le microscope et coregistrées entre elles. Cette méga-image est donc réduite afin de tenir sur une page. On y remarque un cadre rouge qui délimite la zone d'où est tirée la figure précédente. La méga-image représente environ $1cm$ de large (la figure ayant subi une rotation afin de tenir sur une page, c'est sa hauteur donc qui représente environ $1cm$).

FIG. 49 – Résolution maximale de l'objectif 100×

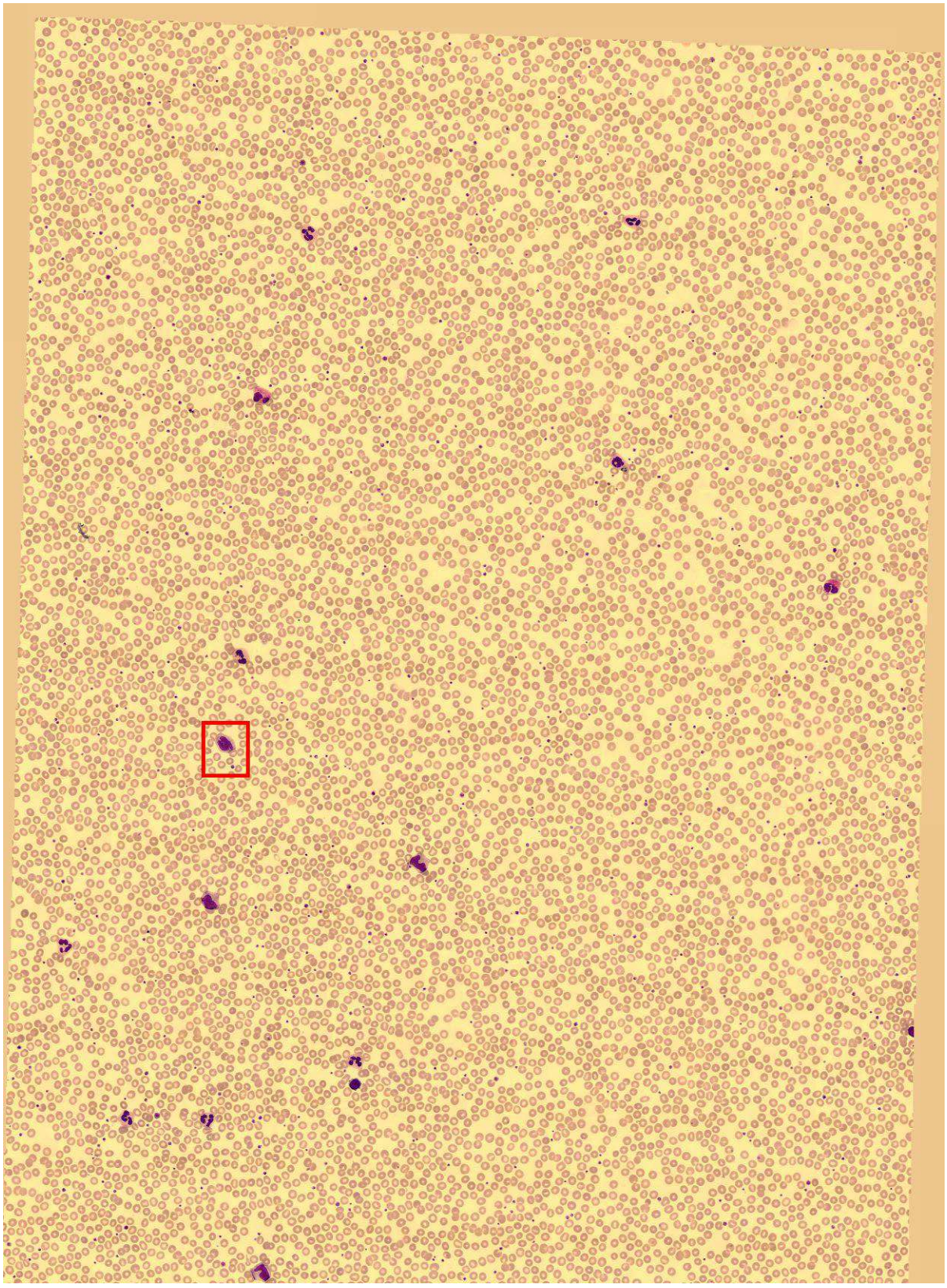

FIG. 50 – Exemple de méga-image de 20 captures de large sur 20 captures de haut

B AnalySIS

Voici une figure représentant l'interface graphique du logiciel AnalySIS Pro 3.0 permettant de visualiser ce qui est observable avec le microscope. C'est via un de ses boutons que l'acquisition d'une méga-image est lancée.

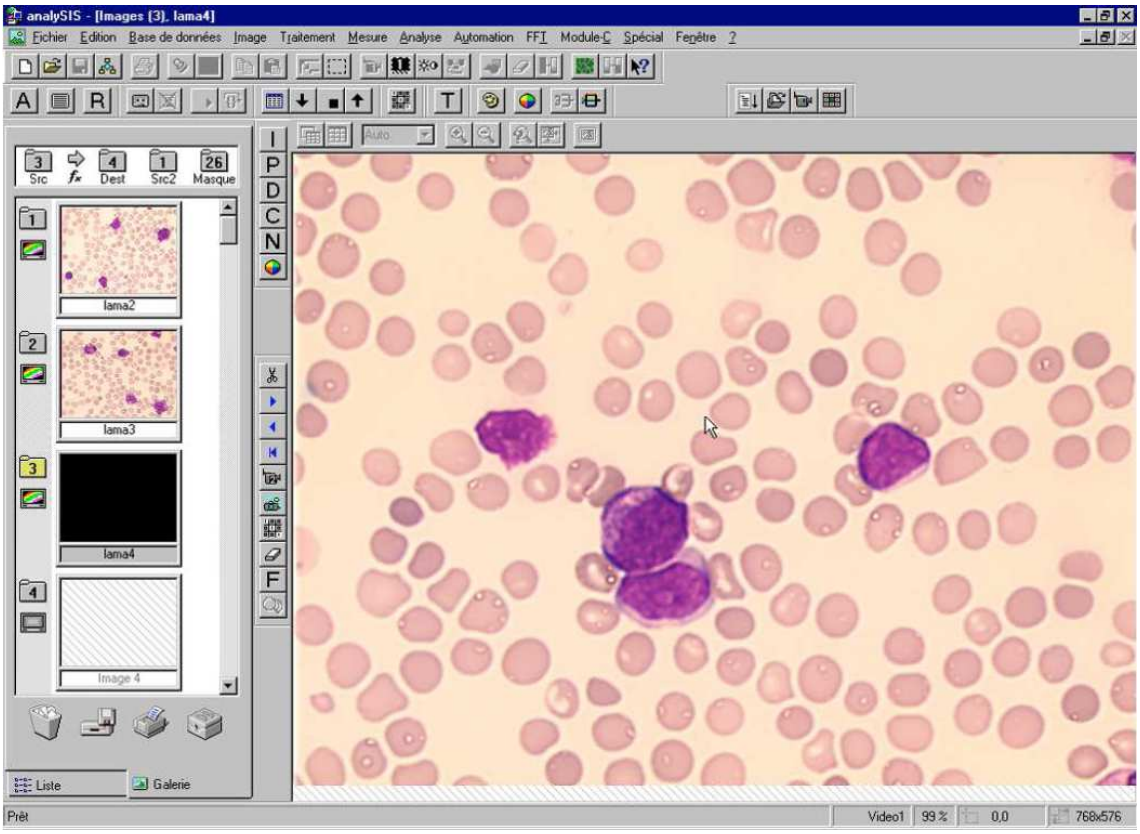

FIG. 51 – Interface du logiciel AnalySIS Pro 3.0

C Problème de calibrage

Il se peut que la coregistration des images, pour former une méga-image, fournisse un résultat incorrect. Un tel résultat peut être observé à la figure 52. Dans ce cas là, il faut recalibrer la caméra fixée sur le microscope à l'aide du logiciel AnalySIS. En effet, si celle-ci est mal calibrée, les images acquises le sont avec un décalage trop important entre chacune d'elles.

Le problème pourrait être évité en modifiant le serveur de création qui crée les méga-images en coregistrant les images capturées. En effet, l'algorithme recherche le bon endroit où joindre les images entre-elles, cependant cette recherche se fait dans une zone limitée de l'image dite image maître. Cette zone limitée ne couvre donc pas le décalage important existant entre les images composantes, ce qui explique qu'aucune jonction correcte n'est trouvée. Le choix de limiter la zone de recherche, pour trouver l'endroit où joindre les images ensemble, est justifié par la performance. Effectivement, si la recherche devait se faire sur toute la hauteur ou la largeur d'une image à chaque fois qu'il y a une image composante à coregistrer, cela ferait perdre beaucoup de temps. Si la caméra est bien calibrée, la coregistration actuelle fonctionne efficacement.

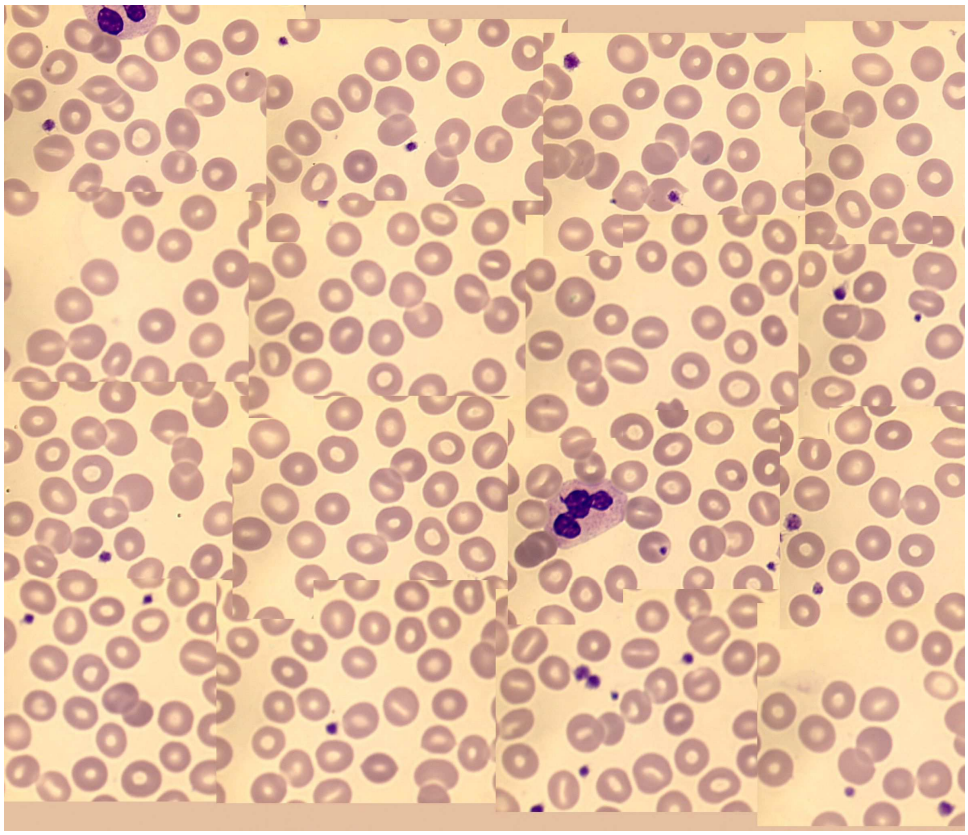

FIG. 52 – Exemple de problème de coregistration

D Lancement du microscope virtuel

Cette annexe fait office de mode d'emploi quant au lancement des divers programmes constituant le microscope virtuel ainsi qu'au lancement du programme d'extraction des images à partir d'une base de données.

D.1 Icône

Avant toute chose, voici l'icône qui a été créée pour le client de visualisation du microscope virtuel. Cette icône remplace ainsi l'éternelle icône représentant la tasse de café si chère à Java, qui est l'icône par défaut de tout programme Java. Notre icône est visible à la figure 53. Le choix du fond orangé est justifié par le fait que l'icône se retrouve aussi bien dans la barre supérieure de la fenêtre de ce dernier que dans la barre des tâches du système d'exploitation. Or la couleur de fond de ces deux emplacements peut ne pas être la même et varie selon les systèmes d'exploitation utilisés, il faut donc une couleur qui se remarque, quelle que soit la couleur de fond de l'emplacement.

FIG. 53 – Icône du microscope virtuel

D.2 Acquisition de méga-images

Afin d'acquérir une méga-image, il faut démarrer le microscope et l'ordinateur relié à ce dernier. Pour démarrer le microscope, il faut procéder comme suit[34] :

1. débrancher la fiche à l'arrière du boîtier U-MCB ;
2. allumer chaque boîtier mis à part celui de l'autofocus (U-AF) ;
3. allumer le microscope ;
4. allumer l'autofocus ;
5. rebrancher la fiche derrière le boîtier U-MCB ;
6. lancer le logiciel AnalySIS sur l'ordinateur démarré, sans cliquer sur "OK" à l'apparition du premier message ;
7. appuyer sur "reset" à l'arrière du boîtier U-MCB ;

[34] Il existe un document expliquant précisément la procédure à suivre dans un classeur à côté du boîtier de contrôle U-MCB.

8. placer la tourelle porte objectifs motorisée du microscope sur l'objectif 10× via le boîtier U-MCB ;
9. enlever la platine du microscope ;
10. cliquer sur "OK" du message affiché par AnalySIS, ceci permet de détecter les butées sur les divers axes du moteur de la platine du microscope ;
11. une fois les butées déterminées automatiquement, replacer la platine, en faisant attention à bien la coincer, avec la lame contenant le frotti numérique ;
12. placer la tourelle porte objectifs sur l'objectif désiré (normalement l'objectif 100×) ;
13. la visualisation avec AnalySIS peut maintenant être démarrée en appuyant sur le bouton en forme de caméra dans l'interface du programme.

Une fois le microscope prêt, il faut démarrer le serveur de création. Ce dernier se présente sous la forme d'un fichier "*.bat*" sur le bureau de l'ordinateur relié au microscope. Si on modifie ce fichier "*.bat*", les paramètres sont :

1. le numéro de port sur lequel le serveur écoute les requêtes de création ;
2. un nombre correspondant au nombre maximum de connexions simultanées que le serveur acceptera ;
3. le paramètre de la machine virtuelle pour étendre sa mémoire : -mx900000000.

Ce serveur de création se sert d'un fichier nommé "*serverCreation.ini*" servant à le paramètrer. Le paramètre qui nous intéresse est nommé "FILE_ROOT" dans la section nommée "[INPUT]". Ce paramètre sert à préciser le répertoire racine où les images, qui serviront à composer la méga-image, seront stockées. Dans ce répertoire racine, les images seront stockées dans un répertoire portant le nom de la méga-image à créer, lui-même stocké dans un répertoire portant le nom de l'utilisateur créant l'image. Une fois la méga-image créée, elle sera stockée dans le répertoire portant le nom de l'utilisateur, toujours dans ce répertoire racine.

Une fois lancé, le serveur de création présente une trace comme suit :

```
img path c:/coreg ... OK

SERVER READY ON PORT : 3200 WAITING FOR CLIENT CONNECTION
```

Le serveur de création lancé, il faut démarrer le client de création via le logiciel AnalySIS. Celui-ci s'occupera de l'acquisition, en pilotant le microscope, des images composant la méga-image. Il se charge aussi d'émettre la requête de création de la méga-image au serveur de création. Le lancement de ce client se fait via un bouton de l'interface de AnalySIS qui affichera une interface demandant les paramètres de la création. Les paramètres importants ici sont le nom de l'utilisateur et le nom de l'image qui permettront de créer les répertoires où stocker les images composantes ainsi que la méga-image créée. Ces répertoires correspondent à ceux que le serveur de création utilisera. Les deux autres paramètres

importants de ce client de création sont le nombre d'images à acquérir qui représenteront la largeur de la méga-image, ainsi que le nombre d'images à acquérir pour la hauteur de cette même méga-image.

Les images composantes sont stockées aux format "*.tif*" tandis que les méga-images, créées à partir de ces images, sont stockées au format "*.j2k*" de JPEG2000. Une méga-image est accompagnée d'un fichier au format "*.idx*, portant le même nom que cette dernière, qui est un fichier d'index facilitant la navigation dans l'image JPEG2000 à l'aide des tiles.

Le client de création envoie une requête de création au serveur de création au format suivant, où le dernier paramètre est facultatif :
$[Version]$ $[User]$ NEW $[MegaImName]$ $[NbrImLarge]$ $[NbrImHaut]$ $[CoefHomogen]$.
Le serveur de création présente ensuite un début de trace comme suit :

```
--> New Client connection from : /127.0.0.1
>> V2.0 cpeeters NEW 1x7 1 7  1.0
<< OK
```

D.3 Visualisation de méga-images

Afin de visualiser une méga-image créée lors de l'étape d'acquisition, il faut premièrement démarrer le serveur de visualisation et deuxièmement démarrer le client de visualisation.

Le serveur de visualisation prend plusieurs paramètres en entrée. Des paramètres initiaux, obligatoires et des paramètres, facultatifs, ajoutés à la suite de ces derniers. Les paramètres initiaux sont :

1. le numéro de port sur lequel le serveur écoute les requêtes de visualisation ;
2. un nombre correspondant au nombre maximum de connexions simultanées que le serveur acceptera ;
3. le chemin du répertoire où sont stockées les méga-images visualisables.

D'autres paramètres ont été ajoutés suite à ce travail. Ces paramètres sont facultatifs et servent en cas de demande d'homogénéisation d'une méga-image (voir point D.4). Ces paramètres ajoutés sont :

1. le numéro de port sur lequel le serveur de création écoute et qui servira à notre serveur de visualisation pour lui envoyer des requêtes d'homogénéisation ;
2. l'adresse où se situe le serveur de création qui permettra l'homogénéisation.

Le serveur de visualisation démarré, le client de visualisation peut s'y connecter. Pour lancer le client de visualisation, il n'y a aucun paramètre à spécifier si ce n'est celui de la machine virtuelle permettant d'augmenter la taille maximale de sa mémoire : -Xmx900000000.

Le client de visualisation débute par une interface permettant la connexion au serveur de création. Il faut donc y entrer l'adresse où se situe le serveur de visualisation ainsi que le port sur lequel il écoute les requêtes. Il faut aussi entrer un nom d'utilisateur et un mot de passe[35]. Le client une fois connecté au serveur, l'interface de visualisation du microscope virtuel se lance.

D.4 Homogénéisation de méga-images

Concernant l'homogénéisation des méga-images, cette dernière peut se faire via l'interface graphique de visualisation. Au moment d'ouvrir une méga-image, il suffit de préciser un coefficient d'homogénéisation. A ce moment, le serveur de visualisation va demander au serveur de création de créer la méga-image homogénéisée sur base des images composantes originales. Si ces images composantes n'existent plus, alors le serveur de visualisation retourne la méga-image originale sans homogénéisation.

Premièrement, il faut que le serveur de création soit démarré. Deuxièmement, il faut démarrer le serveur de visualisation en précisant les paramètres facultatifs qui lui permettront de communiquer avec le serveur de création afin de lui demander de créer des méga-images homogénéisées. Il faut donc que le répertoire où le serveur de création crée les méga-images et que le répertoire où le serveur de visualisation lit les images soient un seul et même répertoire. Ceci afin que l'image créée par le serveur de création puisse être visible par le serveur de visualisation.

Une méga-image homogénéisée est stockée dans un répertoire temporaire nommé "Homogeneisation" et est effacée dès qu'une nouvelle image homogénéisée est demandée. ceci a pour but de ne pas consommer trop d'espace de stockage en créant diverses images homogénéisées. De cette façon, il n'existe qu'une seule image homogénéisée à la fois.

Il ne sera finalement pas possible d'obtenir une méga-image homogénéisée si le serveur de création n'est pas démarré et mis en communication avec le serveur de visualisation. Ou encore si les images composantes, nécessaires à la création de la méga-image homogénéisée ne sont pas présentes.

Il est aussi possible de demander la création d'une image homogénéisée en écrivant un petit programme envoyant des requêtes de création. En voici un exemple :

[35] Pour s'authentifier, on peut utiliser la lettre 't' comme nom d'utilisateur et cette même lettre 't' comme mot de passe. Les couples nom d'utilisateur, mot de passe sont simplement encodés dans un fichier texte nommé "loginpass.txt"

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;

public class TestClient
{
  public static void main(String agrs[])
  {
    try
    {
      Socket clientSocket;
      BufferedWriter outToServer;
      BufferedReader inFromServer;
      clientSocket = new Socket("127.0.0.1", 3200);

      outToServer = new BufferedWriter(new OutputStreamWriter(
          clientSocket.getOutputStream()));
      inFromServer = new BufferedReader(new InputStreamReader(
          clientSocket.getInputStream()));

      //envoyer une requête au serveur de création
      //version user command imageName nbrAbscisse nbrOrdonne
          homogeneisationFactor
      outToServer.write("V2.0 cpeeters NEW 1x7 1 7  1.0"+"\n");

      outToServer.flush();

      System.out.println(inFromServer.readLine().trim());
    }
    catch(Exception e)
    { e.printStackTrace(); }
  }
}
```

D.5 Extraction à partir d'une base de données

L'application d'extraction des images à partir d'une base de données offre la possibilité d'être lancée en mode service ou en mode avec interface graphique. Comme il a été vu dans ce travail, le mode service surveille un répertoire dans l'attente d'une base de données possédant une extension "*.mdb*". Dès qu'une base de données est détectée, ses images en sont extraites et envoyées à l'aide de DICOM. Le mode avec interface graphique ne nécessite aucun paramètre si ce n'est celui pour la machine virtuelle afin d'augmenter la mémoire utilisée : -mx900000000.

Pour lancer le mode service, en plus du paramètre d'extension de la mémoire pour la machine virtuelle, il faut préciser un autre paramètre au programme. Ce paramètre doit contenir le chemin du répertoire que le service doit surveiller. Le service utilise aussi un fichier nommé "*aet.cfg*" pour l'envoi DICOM des images extraites. Au lancement, la trace laissée par le service est la suivante :

```
Waiting for .mdb file in C:/test
```

Le service vérifiera toutes les 30 secondes si une ou plusieurs bases de données sont présentes dans le répertoire placé en paramètre. Si c'est le cas, il les traitera. Pour fonctionner normalement, une base de données doit fournir les tables suivantes, possédant chacune les champs précisés ici :

- **analysis** : *patient, ordered_ at, lab_ id* ;
- **patient** : *id, sex, lab_ id, day_ of_ birth, demographic_ origin, firstname, lastname, ward, medicaldata* ;
- **sample** : *diagnosis* ;
- **diagnosis** : *id, txt* ;
- **classname** : *id, name* ;
- **images** : *id, jpg* ;
- **classification** : *confidence, classname, point, correct.*

E Algorithme de conversion RGB-HSL en Java

Voici, codé en Java[36], comment passer du système RGB au système HSL et inversement :

```
/**
 * Fonction transformant un pixel RGB en HSL (ou TSL en francais
    pour teinte, saturation, luminance)
 * @author Peeters Cédric (algo trouvé sur easyRGB.com)
 * @param rgb<BR>tableau de int contenant les trois composantes R
    , G et B
 * @return double[]<BR>tableau comprenant les trois valeurs H, S
    et L (valeurs comprisent entre 0 et 1)
 */
public double[] RGB2HSL(int rgb[])
{
  double var_R = ( rgb[0] / 255.0 );
  double var_G = ( rgb[1] / 255.0 );
  double var_B = ( rgb[2] / 255.0 );
  //Min. value of RGB
  double var_Min = Math.min( Math.min(var_R, var_G), var_B );
  //Max. value of RGB
  double var_Max = Math.max( Math.max(var_R, var_G), var_B );
  double del_Max = var_Max - var_Min; //Delta RGB value

  double L = ( var_Max + var_Min ) / 2;
  double H = 0, S = 0;

  if ( del_Max == 0 ) //This is a gray, no chroma...
  {
     H = 0;
     S = 0;
  }
  else//Chromatic data...
  {
     if ( L < 0.5 )
     { S = del_Max / ( var_Max + var_Min ); }
     else
     { S = del_Max / ( 2 - var_Max - var_Min ); }

     double del_R=(((var_Max - var_R )/6) + (del_Max/2))/del_Max;
```

[36] Algorithme trouvé sur [S.r]

```
        double del_G=(((var_Max - var_G )/6) + (del_Max/2))/del_Max;
        double del_B=(((var_Max - var_B )/6) + (del_Max/2))/del_Max;

        if( var_R == var_Max )
        { H = del_B - del_G; }
        else if ( var_G == var_Max )
        { H = ( 1 / 3 ) + del_R - del_B; }
        else if ( var_B == var_Max )
        { H = ( 2 / 3 ) + del_G - del_R; }

        if ( H < 0 ){ H += 1; }
        if ( H > 1 ){ H -= 1; }
    }

    double hsl[] = new double[3];
    hsl[0] = H;
    hsl[1] = S;
    hsl[2] = L;

    return hsl;
}

/**
 * Fonction permettant de retrouver les composantes RGB d'un
     pixel en fonction des composantes HSL de celui-ci
 * @author Peeters Cédric (algo trouvé sur easyRGB.com)
 * @param hsl<BR>tableau de double contenant les valeurs de H, S
     et L
 * @return int[]<BR>tableau de int contenant les valeurs de R, G
     et B
 */
public int[] HSL2RGB(double hsl[])
{
    double R = 0, G = 0, B = 0;
    double H = hsl[0];
    double S = hsl[1];
    double L = hsl[2];
    double var_1, var_2;

    if ( S == 0 )
    {
        R = L * 255;
        G = L * 255;
        B = L * 255;
```

```
  }
  else
  {
     if ( L < 0.5 ){ var_2 = L * ( 1 + S ); }
     else { var_2 = ( L + S ) - ( S * L ); }

     var_1 = 2 * L - var_2;

     R = 255 * Hue_2_RGB( var_1, var_2, (H + ( 1.0 / 3.0 )) );
     G = 255 * Hue_2_RGB( var_1, var_2, H );
     B = 255 * Hue_2_RGB( var_1, var_2, (H - ( 1.0 / 3.0 )) );
  }

  int rgb[] = new int[3];

  rgb[0] = (int)Math.round(R);
  rgb[1] = (int)Math.round(G);
  rgb[2] = (int)Math.round(B);

  return rgb;
}

public double Hue_2_RGB(double v1, double v2, double vH )
{
   if ( vH < 0 ) { vH += 1; }
   if ( vH > 1 ) { vH -= 1; }

   if ( ( 6 * vH ) < 1 )
   { return (v1 + (v2 - v1) * 6 * vH); }

   if ( ( 2 * vH ) < 1 )
   { return (v2); }

   if ( ( 3 * vH ) < 2 )
   { return (v1 + (v2 - v1) * ((2.0 / 3.0) - vH) * 6); }
   return ( v1 );
}
```

F Algorithmes de seuillage en Java

Cette annexe présente comment coder en Java, les deux méthodes permettant d'obtenir un seuil utile au seuillage : la méthode d'Otsu et la méthode de l'entropie maximum.

F.1 Méthode d'Otsu

Voici comment déterminer un seuil, en Java, à l'aide de la méthode d'Otsu :

```
int histo[]; //on doit y placer l'histogramme des niveaux de gris
    de l'image

/**
 * Fonction de seuillage d'Otsu
 * @param width<BR>la largeur de l'image qui a servi à créer l'
    histogramme
 * @param height<BR>la hauteur de l'image qui a servi à créer l'
    histogramme
 * @return int contenant la valeur du seuil trouvé avec cette
    technique
 */
public int otsu(int width, int height)
{
        int i,j,seuil=0;
        float sigma[] = new float[256];
        float muT = 0;

        muT = 0;
        for (i=0;i<256;i++)
        { muT += (i+1)*histo[i]; }

        muT /= (width*height);

        for (i=1; i<255; i++)
        { sigma[i] = sig(i,(int)muT,width,height); }

        i=0;
        seuil=0;
        for (j=1; j<255;j++)
        {
                if (i < sigma[j]) /* on a un nouveau maximum */
                {
                  /* on met le seuil dans k */
                  seuil=j; i=(int)sigma[j];
```

```
                }
        }

        return seuil;
}


private float sig(int k, int mu, int taille_x, int taille_y)
{
  float tmp=0, w=0, tmp2;
  float N = (float)(taille_x*taille_y);
  int i;

  for (i=0 ; i<=k; i++)
  {
          w += histo[i];
          tmp += (i+1)*histo[i];    /* on calcul mu(k)*/
  }
  w /= N;
  tmp /= N;
  tmp2 = (mu*w-tmp)*(mu*w-tmp)*(w*(1-w));

  return tmp2;
}
```

F.2 Méthode de l'entropie maximum

Voici comment déterminer un seuil, en Java, à l'aide de la méthode de l'entropie maximum :

```
/**
 * Fonction de seuillage par Maximum Entropy
 * @param histog<BR>int[] contenant l'histogramme de l'image que
    l'on veut seuiller
 * @return un int contenant la valeur du seuil trouvé avec cette
    technique
 */
public int entropySplit(int[] histog)
{
  // Normalize histogram, that is makes the sum of all bins equal
     to 1.
  double sum = 0;
  for (int i = 0; i < histog.length; ++i)
  { sum += histog[i]; }

  double[] normalizedHist = new double[histog.length];
  for (int i = 0; i < histog.length; i++)
  { normalizedHist[i] = histog[i] / sum; }

  double[] pT = new double[histog.length];
  pT[0] = normalizedHist[0];
  for (int i = 1; i < histog.length; i++)
  { pT[i] = pT[i - 1] + normalizedHist[i]; }

  // Entropy for black and white parts of the histogram
  final double epsilon = Double.MIN_VALUE;
  double[] hB = new double[histog.length];
  double[] hW = new double[histog.length];
  for (int t = 0; t < histog.length; t++)
  {
        // Black entropy
        if (pT[t] > epsilon)
        {
          double hhB = 0;
          for (int i = 0; i <= t; i++)
          {
                  if (normalizedHist[i] > epsilon)
                  { hhB -= normalizedHist[i] / pT[t] * Math.log(
                     normalizedHist[i] / pT[t]); }
```

```
            }
            hB[t] = hhB;
          }
          else
          { hB[t] = 0; }

          // White  entropy
          double pTW = 1 - pT[t];
          if (pTW > epsilon)
          {
            double hhW = 0;
            for (int i = t + 1; i < histog.length; ++i)
            {
                  if (normalizedHist[i] > epsilon)
                  { hhW -= normalizedHist[i] / pTW * Math.log(
                      normalizedHist[i] / pTW); }
            }
            hW[t] = hhW;
          }
          else
          { hW[t] = 0; }
    }

     // Find histogram index with maximum entropy
    double jMax = hB[0] + hW[0];
    int tMax = 0;
    for (int t = 1; t < histog.length; ++t)
    {
          double j = hB[t] + hW[t];
          if (j > jMax)
          {
            jMax = j;
            tMax = t;
          }
    }

    return tMax;
}
```

G Algorithme de filtrage médian en Java

Un filtre médian trie en ordre croissant les valeurs des pixels voisins d'un pixel et remplace la valeur de ce pixel par la médiane du tri. Voici comment faire cela en Java :

```
/**************************************
 * @author Peeters Cédric
 * Fonction appliquant un filtre median
 * sur l'image reçue en paramètre
 */
private void median(BufferedImage bufImage)
{
  int w = width;
  int h = height;

  //quadrillage pour un voisinage de 3x3 pixels
  int[] squareShape = new int[15];

  for (int v=1; v<=h-2; v++)
  {
    for (int u=1; u<=w-2; u++)
    {
      //remplir le quadrillage pour la position (u,v)
      int k = 0;
      for (int j=-1; j<=1; j++)
      {
        for (int i=-1; i<=1; i++)
        {
          squareShape[k] = bufImage.getRGB(u+i,v+j);
          k++;
        }
      }
      //trier le quadrillage et en prendre l'élément central
      Arrays.sort(squareShape);
      bufImage.setRGB(u,v,squareShape[4]);
    }
  }
}
```

H Manipulation de fichiers MS-Excel en Java

Pour créer et manipuler des fichiers Excel avec l'extension "*.xls*" en Java, c'est le package *jxl*[37] qui a été utilisé. La version actuelle de ce package est codée en Java 1.5.

Voici comment lire dans un fichier "*.xls*" :

```
import jxl.* ;

...

// déclarer le fichier Excel à lire
Workbook fichierExcel = Workbook.getWorkbook(new File("monFichier
    .xls"));

// déclarer la feuille à lire dans le fichier
// la numérotation des feuilles commence à 0 pour la 1ère feuille
     et ainsi de suite
Sheet feuille = fichierExcel.getSheet(0);
double nombre = 0;

//déclarer la cellule à lire
//la numérotation des cellules commence à (0,0) => (lettre de la
    colonne, chiffre de la ligne)
Cell a1 = feuille.getCell(0,0);

//vérifier si le format de la cellule est bien numérique
if (a1.getType() == CellType.NUMBER)
{
  //déclarer une cellule de type numérique
  NumberCell celulleNumerique = (NumberCell) a1;
  //placer la valeur de cette cellule dans la variable
  nombre = celulleNumerique.getValue();
}

fichierExcel.close();
```

[37] Package disponible sur `http://jxl.sourceforge.net/`. et son API est disponible sur `http://www.andykhan.com/jexcelapi/`

Voici comment écrire dans un fichier "*.xls*" :

```
import jxl.write.* ;

...

// déclarer le fichier Excel à créer
WritableWorkbook fichierExcel = Workbook.createWorkbook(new File(
    "monFichier.xls"));

// déclarer la feuille à créer dans le fichier
// la numérotation des feuilles commence à 0 pour la 1ère feuille
     et ainsi de suite
WritableSheet feuille = fichierExcel.createSheet("Feuille1", 0);
double nombre = 0;

//déclarer la cellule à écrire et l'ajouter à la feuille
//la numérotation des cellules commence à (0,0) => (lettre de la
    colonne, chiffre de la ligne)
Number nombre = new Number(3, 4, 3.1459); // new Number(lettre,
    chiffre, nombre à écrire)
feuille.addCell(nombre);

// écrire le fichier et le fermer
fichierExcel.write();
fichierExcel.close();
```

Voici comment insérer une image dans une cellule[38] (l'image prendra la taille de la cellule et non l'inverse) :

```
WritableImage celluleImage = null;
WritableWorkbook workbook = null;
WritableSheet sheet = null;

try
{
  // créer le fichier
  workbook = Workbook.createWorkbook(new File(nomImage+".xls"));
  // créer la 1ère feuille (feuille 0) dans le fichier
  sheet = workbook.createSheet("imagesEtParametres", 0);
}
catch (IOException e)
{ e.printStackTrace(); }

try
{
  // placer l'image dans une cellule
  // colonne 1, ligne 2, largeur de 1 colonne, hauteur de 1 ligne
  celluleImage = new WritableImage(1, 2, 1, 1,new File("
      repertoireImage","nomImageAvecExtension"));
  // ajouter la cellule à la feuille
  sheet.addImage(celluleImage);
}
catch (RowsExceededException e)
{ e.printStackTrace(); }
catch (WriteException e)
{ e.printStackTrace(); }

try
{
  workbook.write();
  workbook.close();
}
catch (IOException e1)
{ e1.printStackTrace(); }
catch (WriteException e)
{ e.printStackTrace(); }
```

[38] Dans l'application du client de visualisation, la création des fichiers Excel pour les globules blancs se trouve dans la classe *WhiteGlobImageDetector* dans le package *tools*.

Ce package a servi pour la création de fichiers Excel afin de pouvoir trier les globules blancs selon divers critères. Voici un exemple de fichier Excel obtenu après création d'une galerie d'images centrées sur les globules blancs :

	A	B	C	D	E	F
1	Nom image	Image	Surface globule (en pixels)	Surface carré entourant globule (en pixels)	Taux remplissage (en %)	Rapport Largeur/Hauteur du carré
2	8x3Im01.png		10294	20020	51,41858142	1,184615385
3	8x3Im02.png		9756	16819	58,00582674	1,148760331
4	8x3Im03.png		9456	17812	53,08780597	0,835616438

FIG. 54 – Fichier Excel de classification

I Astuces Java

Cette annexe présente quelques astuces Java qui ont servi lors du développement du microscope virtuel.

I.1 Modification d'un *JFileChooser*

Voici comment faire pour modifier les dénominations des divers éléments d'un *JFileChooser*. Ceci a permis de traduire le *JFileChooser* du français vers l'anglais puisque le reste de l'application de visualisation est en anglais.

Les textes affichés dans un *JFileChooser* dépendent de ressources du *UIManager*. Il faut donc modifier ces ressources avant de construire le *JFileChooser*. Ceci se fait comme suit :

```
UIManager.put("FileChooser.fileNameLabelText", "Gallery name");
UIManager.put("FileChooser.saveButtonText", "Save");
UIManager.put("FileChooser.cancelButtonText", "Cancel");
UIManager.put("FileChooser.cancelButtonToolTipText","Cancel
    saving");
UIManager.put("FileChooser.upFolderToolTipText", "Upper level");
UIManager.put("FileChooser.homeFolderToolTipText", "Home");
UIManager.put("FileChooser.newFolderToolTipText", "Create new
    folder");
UIManager.put("FileChooser.listViewButtonToolTipText", "List");
UIManager.put("FileChooser.detailsViewButtonToolTipText", "
    Details");
UIManager.put("FileChooser.filesOfTypeLabelText", "Type of file")
    ;
UIManager.put("FileChooser.lookInLabelText", "Look in");
UIManager.put("FileChooser.acceptAllFileFilterText","All file");

JFileChooser chooser = new JFileChooser();
```

I.2 Obtenir le type d'un fichier

Voici comment faire pour connaître le type MIME d'un fichier. Cette méthode a été utile pour repérer les fichiers au format *Tiff*. En effet, ce format peut être écrit de diverses manières, en vérifiant le type MIME, il n'y en a qu'un seul de possible.

```
File fichier = new File("nomFichier");
// obtenir le type du fichier
String typeFichier = fichier.toURL().openConnection().
    getContentType();


// on peut maintenant tester le type du fichier
if(typeFichier.endsWith("image/tiff"))
{

}
```

I.3 Obtenir la fenêtre active

Dans l'application de visualisation, les fenêtres de message, d'ouverture de fichier ou encore de visualisation d'une image de globule blanc en pleine résolution sont toutes des *JWindow*. Il est possible de lier une *JWindow* à la fenêtre active. Dans notre cas, la fenêtre active est celle du visualisateur des méga-images. En liant une *JWindow* à la fenêtre active, cela évite d'avoir deux fenêtres indépendantes. Voici comment obtenir la fenêtre active et lier une *JWindow* à cette dernière :

```
// obtenir la fenêtre active
Window activeWindow = KeyboardFocusManager.
    getCurrentKeyboardFocusManager().getActiveWindow();

//créer une fenêtre sans barre de titre, liée à la fenêtre active
JWindow window = new JWindow(activeWindow);
```

I.4 Monitorer la mémoire

Voici comment observer la mémoire utilisée par un programme Java :

```
Runtime runtime = Runtime.getRuntime();
System.out.println("Memory");

System.out.println("used: " + (runtime.totalMemory() - runtime.
    freeMemory()));
System.out.println("committed: " + runtime.totalMemory());
System.out.println("max: " + runtime.maxMemory());
```

J Diagrammes de classes

Cette annexe présente un diagramme de classes simplifié pour l'extracteur d'images à partir d'une base de données ainsi qu'un diagramme simplifié pour la création de galeries d'images de globules blancs.

Voici, à la figure 55 le diagramme de classes correspondant à l'application d'extraction des images à partir d'une base de données. Le programme se lance soit en mode "service", soit en mode avec interface graphique. La classe *Extractor* est appelée afin d'extraire les images de la base de données. Cette classe, dans le cas où le mode "service" est lancé, peut faire appel à la classe *DICOMSender* pour envoyer les images à l'aide du standard DICOM. Au niveau de l'interface graphique, on constate qu'elle est composée d'une galerie d'images et d'un compteur de globules.

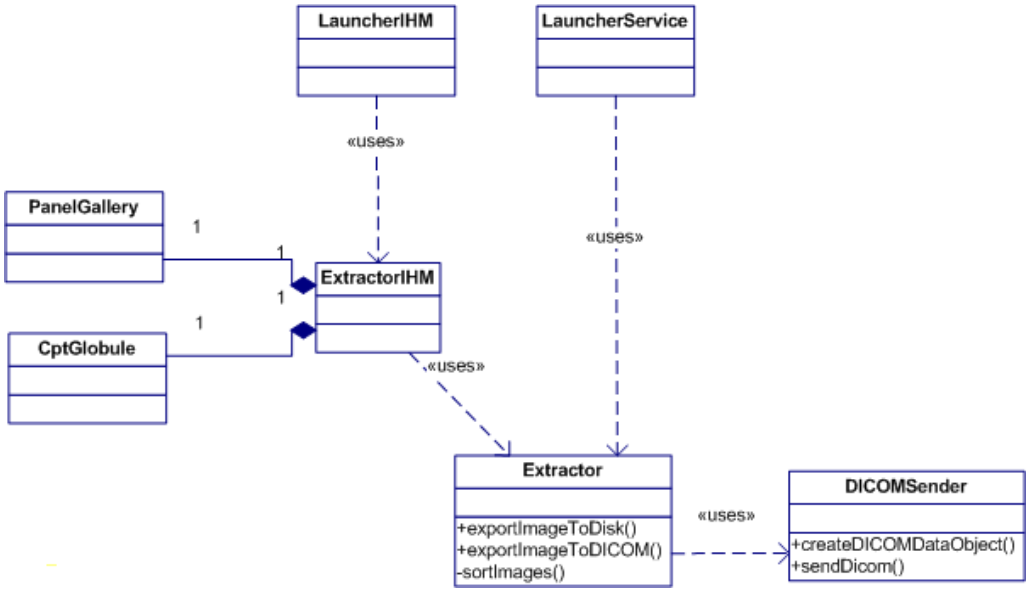

FIG. 55 – Diagramme de classes de l'extracteur d'images

En ce qui concerne le visualisateur de méga-image, permettant la création d'une galerie d'images de globules blancs, créer un diagramme de classes complet demanderait un travail de rétro-ingénierie complet pour documenter les travaux des stagiaires précédents. Le diagramme présent à la figure 56 illustre uniquement les classes permettant la création d'une galerie d'images et ne présente donc qu'une infime partie du diagramme complet de l'application. On s'aperçoit dans ce diagramme que la fenêtre principale de l'interface graphique est composée d'un panneau permettant l'affichage d'une galerie. Lorsqu'on clique sur le bouton initiant la création d'une galerie, c'est la classe *WhiteGlobImageDetector* qui est appelée. Cette classe étant un thread, le reste de l'application peut toujours fonctionner tandis que la galerie est en cours de création. Cette classe détecte les tiles contenant des zones de globules blancs en faisant appel aux techniques de seuillage et lance un autre thread. Ce dernier correspond à la classe *WhiteGlobZoneDetector* qui détecte, parmi les tiles reçues précédemment, les zones qui correspondent à des globules blancs. La détection se fait en utilisant les divers traitements d'image décrits durant ce mémoire. Une fois les zones de globules détectées, c'est à nouveau la classe *WhiteGlobImageDetector* qui s'occupe de créer les images qui représentent les globules détectés, et les affiche dans le panneau de l'interface graphique de la galerie via la fonction *addImageToGallery()*.

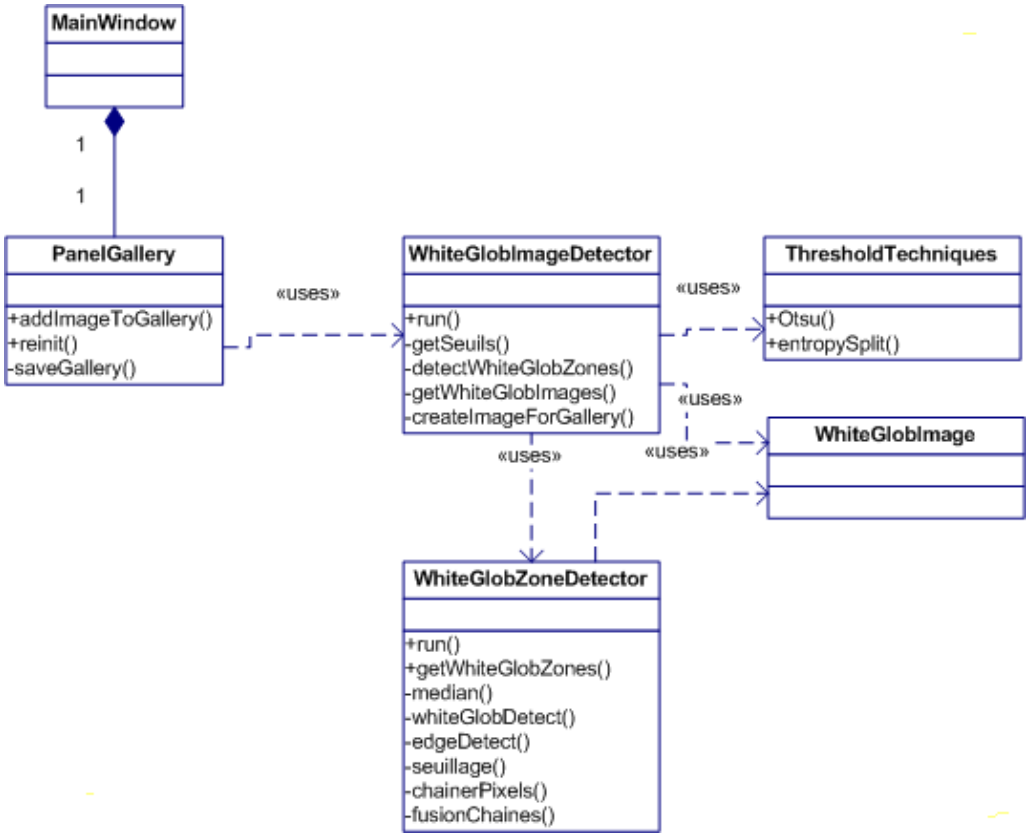

FIG. 56 – Diagramme de classes de la création d'une galerie

Finalement, l'application d'homogénéisation d'image ne fait pas l'objet d'un diagramme de classes étant donné qu'elle consiste en la modification de classes déjà existantes du serveur de création. Les classes qui ont été modifiées sont la classe *SessionManager* et la classe *DecoderTiff*. La classe *SessionManager* crée les images homogénéisées, si une homogénéisation est demandée, afin de créer une méga-image. La classe *DecoderTiff* récupérera les images homogénéisées plutôt que les images d'origine lors de leur réquisition pour créer la méga-image.

Références bibliographiques

b

Références bibliographiques

Bibliographie

[Ang03] J. Angulo. Simplification morphologique d'images couleur par critères connectifs. *Ph.D. Thesis Ecole des Mines de Paris*, 2003.

[BL04] M. Beaudouin-Lafon. Traitement d'images. *Informatique Graphique*, 2004.

[Boy04] K. Boyer. Computer vision : Segmentation. *Cours d'imagerie de l'Ohio State University*, 2004.

[cL99] Olympus Optical co LTD. Système d'autofocus pour microscopes d'emploi général (dispositif de mise au point motorisée utilisant la méthode U-ZD). *Mode d'emploi U-AF*, 1999.

[Cor04] M. Cord. Traitement d'images : application à la reconstruction de surfaces et à la recherche interactive d'images. *Traitement de l'Image et du Signal à l'Universit ´e de Cergy-Pontoise*, 2004.

[Cro06] J. Crowley. Vision par ordinateur : Reconnaissance de formes dans les images. *DEA IVR, Premier Bimestre 2005/2006*, 2006.

[CVB97] S. Muller C. Vachier and S. Bothorel. Analyse morphologique des clichés mamographiques en vue de la détection des opacités du sein. *Proc. of ADEBIO*, 1997.

[DBR06] B. Parrein D. Barba, P. Le Callet and V. Ricordel. Traitement numérique d'images et codage. *Cours de l'université numérique ingénierie et technologie*, 2006.

[DD02] H. DEITEL and P. DEITEL. *Comment programmer en JAVA*. 4ème édition, Les éditions Reynald Goulet Inc., Repentigny (Québec), 2002.

[Dec04] G. Decock. Composition automatique de frottis numériques - application aux gigaimages. *Mémoire de fin d'études en informatique, FUNDP(Facultés Universitaires Notre- Dame de la Paix) Namur Belgique*, 2004.

[Fla01] D. Flanagan. *Exemples en Java in a nutshell.* 2ème édition, Editions O'Reily, Paris, 2001.

[Geo02] B. Georges. La télémicroscopie en cytologie hématologie. *Mémoire de fin d'études en informatique, FUNDP(Facultés Universitaires Notre- Dame de la Paix) Namur Belgique*, 2002.

[Gir87] G. Giraudon. Chaînage efficace de contours. *Rapport de recherche n°605, Institut National de Recherche en Informatique et en Automatique*, 1987.

[GM02] V. Gouet and P. Montesinos. Normalisation des images en couleur face aux changements d'illumination. In *13ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle*, volume II, pages 415–424, Angers, France, 2002.

[Jad05] C. Jadoul. Imagerie Médicale - Etude et développement d'un microscope virtuel. *Mémoire de fin d'études en informatique, FUNDP(Facultés Universitaires Notre-Dame de la Paix) Namur Belgique*, 2005.

[Oly05] Olympus. Informations complémentaires U-IFRS/U-IFGP, carte RS232C et carte GPBI. *Manuel d'utilisation du pupitre multi-commandes U-MCB*, 2005.

[Ots79] N. Otsu. A threshold selection method from gray level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 1979.

[Per06] B. Permentier. Etude et développement d'un système de télémicroscopie virtuelle. *Mémoire de fin d'études en informatique, IESN(Institut d'Enseignement Supérieur de Namur) Namur Belgique*, 2006.

[PH74] T. Pavlidis and S. Horowitz. Segmentation of plane curves. *IEEE. Trans. Computers*, 1974.

[PV00] C. Potier and C. Vercken. Qu'est-ce que la correction gamma ? *Cours pour le Mastère Multimédia de l'Ecole nationale supérieure des télécommunications à Paris*, 2000.

[Ric00] S. Richetto. Validation du logiciel mustig pour la simulation et l'implémentation d'algorithmes de traitement d'images. *Dea SIPT IEG*, 2000.

[Sav05] B. Savelli. Analyse d'image. *Formation en Microscopie Imagerie*, 2005.

[SS01] L. Shapiro and G. Stockman. *Computer Vision*. Prentice Hall, 2001.

[SS04] B. Sankur and M. Sezgin. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 2004.

[Sta04] L. Starzack. Traitement d'image (tome 1 et 2). *Cours de traitement d'image, INPRES (Institut Provincial d'Enseignement Supérieur) Haute Ecole de la Province de Liège Rennequin Sualem, Seraing Belgique*, 2004.

[Zuy03] L. Zuyderhoff. Composition automatique de frottis numériques. *Mémoire de fin d'études en informatique, FUNDP(Facultés Universitaires Notre- Dame de la Paix) Namur Belgique*, 2003.

Netographie

[Bou] P. Bourke. *HSL colour space.* (Dernière visite le 25 octobre 2006), http ://local.wasp.uwa.edu.au/ pbourke/colour/hsl/.

[Bou05] C. Boudry. *Segmentation des images.* (Dernière visite le 27 octobre 2006), http ://www.ext.upmc.fr/urfist/image_numerique/segmentation.htm, 2005.

[Boy04] K. Boyer. *Otsu's Thresholding Method.* (Dernière visite le 27 octobre 2006), http ://sampl.ece.ohio-state.edu/EE863/2004/ECE863-G-segclust2.ppt, 2004.

[cdt06] Centre canadien de télédétection. *Analyse et interprétation d'images.* (Dernière visite le 31 octobre 2006), http ://ccrs.nrcan.gc.ca/resource/tutor/fundam/chapter4/04_f.php, 2006.

[Cha02] P. Chan. *The Java Developers Almanac 1.4.* (Dernière visite le 30 novembre 2006), http ://www.exampledepot.com/egs/javax.swing.filechooser/DoneEvent.html, 2002.

[coma] commentcamarche.net. *Codage HSL (STL).* (Dernière visite le 18 octobre 2006), http ://www.commentcamarche.net/video/hsl-tsl.php3.

[comb] commentcamarche.net. *Introduction au traitement d'images.* (Dernière visite le 17 octobre 2006), http ://www.commentcamarche.net/video/traitimg.php3.

[Cov06] L. Cova. *Eliminer les fuites mémoires.* (Dernière visite le 29 novembre 2006), http ://louis.cova.neuf.fr/blocs-notes/page2.html, 2006.

[DA04] M. Davidson and M. Abramowitz. *Basic Concepts in Digital Image Processing.* (Dernière visite le 12 décembre 2006), http ://microscopy.fsu.edu/primer/digitalimaging/javaindex.html, 2004.

[dCeB06] Institut Européen de Chimie et Biologie. *Définition générale de la leucémie myeloïde chronique.* (Dernière visite le 19 décembre 2006), http ://www.cellbiol.net/GleevecGroupe1.html, 2006.

[dev05] developpez.net. *Comment connaitre la mémoire utilisée par notre application pendant son exécution ?* (Dernière visite le 28 novembre 2006), http ://www.developpez.net/forums/showthread.php ?t=2032&page=4, 2005.

[DM01] J. Darbon and P. Musé. *Segmentation en régions : l'histogramme d'homogénéité.* (Dernière visite le 27 octobre 2006), http ://www.tsi.enst.fr/tsi/enseignement/ressources/mti/Cheng/, 2001.

[dt06] Centre d'information topographique. *Centre de gravité d'une surface plane.* (Dernière visite le 09 novembre 2006), http ://www.cits.rncan.gc.ca/cit/servlet/CIT/site_id=01&page_id=2-007-002-007.html, 2006.

[Ent06] Jelsoft Enterprises. *Fonction luminance.* (Dernière visite le 18 octobre 2006), http ://forum.games-creators.org/archive/index.php/t-2856.html, 2006.

[Fau01a] J. Fauqueur. *Traitement et Manipulation d'image.* (Dernière visite le 13 septembre 2006),
http ://www-rocq.inria.fr/who/Julien.Fauqueur/java/sujet/projet.html, 2001.

[Fau01b] J. Fauqueur. *Traitement et Manipulation d'image.* (Dernière visite le 24 octobre 2006),
http ://docs.ufrmd.dauphine.fr/java/projets/manip_images/projet.html, 2001.

[Fla] A. Flamand. *Correction Gamma avec GammaRight.* (Dernière visite le 17 octobre 2006),
http ://alexflam.free.fr/gamma.htm.

[GRE] GREYC. *Conversion.* (Dernière visite le 22 septembre 2006),
http ://www.greyc.ensicaen.fr/EquipeImage/Pandore/programmes/operatorsP0.html.

[Hen05] N. Hendrich. *Image processing : Sobel filter.* (Dernière visite le 14 septembre 2006),
http ://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/00-intro/02-imageprocessing/sobel.html, 2005.

[Hov00] M-A. Van Hove. *Formulaires.* (Dernière visite le 24 octobre 2006),
http ://www.sc.ucl.ac.be/ete-mathphys/droite/Droite.htm, 2000.

[Hux] J. Huxtable. *Java Image Processing.* (Dernière visite le 17 octobre 2006),
http ://www.jhlabs.com/ip/filters/index.html.

[ima] images.ciel. *Réaliser une mosaïque lunaire.* (Dernière visite le 17 octobre 2006),
http ://images.ciel.free.fr/Fversion/Lune/Realisation.html.

[Jan04] N. Janey. *Le traitement d'images.* (Dernière visite le 18 octobre 2006),
http ://raphaello.univ-fcomte.fr/IG/TraitementImages/TraitementImages.htm#luminosite, 2004.

[JO06] F. Jean and J-N. Ouellet. *Vision numérique.* (Dernière visite le 27 octobre 2006),
http ://wcours.gel.ulaval.ca/2006/a/19263/default/7references/index.chtml, 2006.

[JS06a] C. Jollivet and E. Siber. *FAQ Java GUI, JFileChooser.* (Dernière visite le 29 novembre 2006),
http ://www.exampledepot.com/egs/javax.swing.filechooser/DoneEvent.html, 2006.

[JS06b] C. Jollivet and E. Siber. *FAQ Java GUI, les images.* (Dernière visite le 13 septembre 2006),
http ://java.developpez.com/faq/java/ ?page=graphique_general_images, 2006.

[Kod] Koders. *Searching 631,768,147 lines of code.* (Dernière visite le 20 septembre 2006),
http ://www.koders.com/.

[Lem06] C. Lemmel. *HematoVision.* (Dernière visite le 19 décembre 2006),
http ://www.opus-species.com/hematovision.html, 2006.

[LM06] D. Lingrand and J. Montagnat. *Image Analysis and Processing (IS4).* (Dernière visite le 20 septembre 2006), http ://www.essi.fr/ lingrand/Ens/ImageProcessingCourse.html, 2006.

[Lux] Luxorion. *La restitution des images sur ordinateur.* (Dernière visite le 24 octobre 2006), http ://www.astrosurf.com/luxorion/rapport-restitution-images-ordinateur.htm.

[Man06] A. Manzanera. *Traitement d'images et vision artificielle.* (Dernière visite le 27 octobre 2006), http ://www.ensta.fr/ manzaner/Cours/D9_1/, 2006.

[mat] mathtools.net. *Image Processing.* (Dernière visite le 17 octobre 2006), http ://www.mathtools.net/Java/Image_Processing/.

[Mat06] Color Matters. *Computer Color Matters.* (Dernière visite le 17 octobre 2006), http ://www.colormatters.com/comput.html, 2006.

[McG03] J. McGowan. *Image Processing in Java.* (Dernière visite le 20 septembre 2006), http ://www.jmcgowan.com/sword.html, 2003.

[Mon] P. Monger. *HSL and RGB conversion.* (Dernière visite le 25 octobre 2006), http ://130.113.54.154/ monger/hsl-rgb.html.

[Sch03] M. Schulze. *Image Processing in Java.* (Dernière visite le 13 septembre 2006), http ://www.markschulze.net/java/index.html, 2003.

[SMB03] M. Nogaj S. Makni and J. Battais. *Segmentation d'images par ligne de partage des eaux sous contraintes.* (Dernière visite le 13 septembre 2006), http ://www.tsi.enst.fr/tsi/enseignement/ressources/mti/lpe/index.html, 2003.

[Sof06] Helicon Soft. *Comment utiliser l'histogramme.* (Dernière visite le 24 octobre 2006), http ://helicon.com.ua/filter/help/french/how_to_use_histogram.html, 2006.

[S.r] Logicol S.r.l. *Color Conversion Formulas.* (Dernière visite le 18 octobre 2006), http ://www.easyrgb.com/math.php ?MATH=M18#text18.

[Tel] Telesun. *Exemple de seuillage par détection de vallées.* (Dernière visite le 27 octobre 2006), http ://telesun.insa-lyon.fr/ telesun/Analyse/L02/exemple.html.

[TGD01] P-Y. Strub T. Géraud and J. Darbon. *Segmentation d'Images en Couleur par Classification Morphologique Non Supervisée.* (Dernière visite le 13 septembre 2006), http ://www.lrde.epita.fr/dload/papers/icisp01-article/index.html, 2001.