



UNIVERSITÉ DE BORDEAUX

INITIATION À LA RECHERCHE
ET/OU DÉVELOPPEMENT

Rapport du projet de programmation :

Développement d'un utilitaire de sélection de
particules observées au microscope
électronique

Thomas FAUX
Charlotte HÉRICÉ
Typhaine PAYSAN-LAFOSSE
Joris SANSEN
M. Jean-Christophe TAVEAU
2011-2012



CBMN - UMR 5248
B8 AVENUE DES FACULTÉS
F-33402 TALENCE CEDEX

Résumé

Dans le cadre des recherches de l'équipe ACMPC (Architecture des Complexes Membranaires et Processus Cellulaires) de l'unité mixte de recherche CNRS CBMN (Chimie et Biologie des Membranes et Nanoobjets), il est fréquent d'avoir à piquer manuellement un grand nombre de particules (complexes protéiques ou chimiques) sur des micrographies de microscopie électronique. Celles-ci sont ensuite traitées avec des outils statistiques en vue de calculer des images moyennes ou des reconstructions 3D (3 Dimensions). Pour cela, le logiciel ImageJ est couramment utilisé pour cette étape préliminaire, mais il ne dispose d'aucun outil automatique adapté pour traiter une grande quantité de données.

L'objectif de notre projet a donc été de pallier à ce manque en créant un plugin pour ce logiciel. ImageJ étant implémenté en Java™, nous avons créé une interface facile à utiliser en suivant ce langage. A l'heure actuelle, trois méthodes de sélection de particules (piquage) ainsi qu'un outil de découpage (*crop* en anglais) de ces objets ont été implémenté.

Ce plugin, nommé *Pick_EM* a également été créé dans le but de réunir dans un même outils plusieurs algorithmes, chacun ayant une sensibilité différente en fonction des cibles et de la qualité des images. Le code suit donc les règles de la programmation orientée objet afin de faciliter l'insertion d'une nouvelle méthode de piquage.

Remerciements

Nous tenons tout d'abord à remercier Jean-Christophe TAVEAU, notre client et tuteur, et Marie BEURTON-AIMAR pour nous avoir encadré et conseillé tout au long de ce projet, ainsi que d'avoir su nous faire partager leur expérience.

Nous remercions également Olivier LAMBERT de nous avoir accueilli au sein de l'équipe Architecture des Complexes Membranaires et Processus Cellulaires du CBMN.

Nous tenons aussi à remercier Grégoire NAUDIN pour ses conseils, son soutien, son humour et les nombreuses discussions enrichissantes que nous avons partagé.

Table des matières

Introduction	5
1 Analyse	6
1.1 Contexte	6
1.2 État de l'existant	7
1.3 Analyse des besoins	9
1.4 Scénario d'utilisation du plugin	10
2 Conception	11
2.1 Interface graphique	11
2.2 Récupération des paramètres	13
2.3 Algorithmes	13
2.4 Pile d'images résultantes	14
2.5 Organisation orientée objet du programme	14
3 Réalisation	17
3.1 Interface graphique	17
3.2 Récupération des paramètres	21
3.3 Algorithmes	21
3.4 Autres classes	23
3.5 Applications	24
3.6 Ajout d'un nouvel algorithme au plugin	26
3.7 Améliorations possibles	27
Conclusion	28
Bibliographie	28
A Annexes	30
Annexes	30
A.1 Diagramme des classes du plugin Pick_EM	31
A.2 Code source du plugin ImageJ Pick_EM	32

Introduction

Notre projet s'est déroulé au sein du laboratoire de Chimie et Biologie des Membranes et Nanoobjets de Bordeaux (CBMN [1]). Il s'agit d'un laboratoire de recherche public composé de douze équipes de recherche dont l'équipe *Architecture des Complexes Membranaires et processus cellulaires* (ACMPC). Cette équipe, dirigée par O.LAMBERT, s'intéresse à l'architecture de complexes membranaires sur des structures de type protéine-liposome. C'est dans ce cadre d'étude que les chercheurs travaillent avec un *microscope électronique à transmission* (MET) afin d'obtenir des micrographies des structures protéiques puis de les analyser.

Bien que les principaux projets de recherche de cette équipe portent essentiellement sur l'étude des protéines membranaires et des complexes chimiques nanométriques, l'équipe a aussi une activité de services avec par exemple, des projets d'imagerie sur des virus fournis par une entreprise pharmaceutique. Il est donc nécessaire d'automatiser au maximum les différentes étapes de traitement et d'analyse d'images pour avoir un résultat rapide en réduisant les interventions manuelles d'un opérateur (microscopiste, biologiste, etc.). Ceci est d'autant plus vrai que la nouvelle génération de MET permet une collecte de données à haut débit, avantage non négligeable mais qui pose le problème du temps de traitement des données collectées.

Dans le cadre de leurs recherches et pour la partie qui nous intéresse, l'analyse concerne le traitement des images collectées au MET et dont la première étape nommée *picking*, en anglais ; dans ce texte, nous utiliserons le terme "piquage" consiste à isoler chacune des particules dans des petites images. Notre objectif était l'implémentation d'une interface contenant plusieurs méthodes de piquage automatisées. Celle-ci est implémentée en Java™¹ sous la forme d'un plugin ImageJ [3] proposant à l'utilisateur le choix entre différents algorithmes de picking pré-installés. Il pourra en être ajouté de nouveaux si on le souhaite, tout en minimisant le code source du plugin à modifier.

Dans la suite de ce rapport, nous développerons plusieurs points. Tout d'abord une partie Analyse dans laquelle nous remettrons notre projet dans son contexte et ferons un état des lieux des besoins associés. Ensuite, une partie Conception dans laquelle nous expliquerons l'organisation de notre code et enfin, la partie Réalisation contiendra les solutions apportées au sujet.

1. Java™ est un langage orienté-objet développé par Oracle [2]

1.1 Contexte

1.1.1 Présentation du CBMN

Le laboratoire, créé en janvier 2007, est à l'interface entre la Biologie, la Chimie et la Physique. Il a pour tutelle les départements de Chimie et des Sciences de la Vie du Centre National de la Recherche Scientifique (CNRS) et les Unités de Formation et de Recherche (UFR) de Chimie et de Biologie de l'Université Bordeaux I, II et l'ENITAB.

La mission du CBMN est d'apporter une connaissance fondamentale de phénomènes biologiques complexes en les analysant à plusieurs échelles, allant de la molécule à la cellule et à l'organisme.

L'équipe ACMPC s'appuie sur deux méthodes d'imagerie : la cryotomographie électronique et la microscopie à fluorescence. Comme nous l'avons expliqué précédemment, nous allons travailler sur des images issues de cryomicroscopie.

1.1.2 Objectifs

Notre objectif était de créer et d'implémenter une plateforme mettant à disposition plusieurs méthodes de piquage automatisées des particules sur les micrographies. Les échantillons qui nous ont servi de tests étaient de deux types :

- des micrographies d'échantillons de virus, de forme circulaire, qu'il fallait sélectionner afin de pouvoir déterminer leurs nombre et tailles,
- des micrographies de protéines membranaires, de forme pyramidale, qu'il fallait aussi sélectionner.

Il était imposé que cette interface soit implémentée sous la forme d'un plugin ImageJ. Elle propose à l'utilisateur de se servir des algorithmes de piquage pré-installés.

Le logiciel ImageJ a été choisi car c'est un logiciel de traitement et d'analyse d'images développé en Java™ par le National Institute of Health (NIH).

Ce logiciel est "open-source"¹, multi-plateforme et bien connu de la communauté scientifique car initialement conçu pour des applications biomédicales. Il s'est peu à peu démocratisé dans d'autres domaines pour sa facilité d'utilisation et les possibilités de développement qu'il offre.

En effet, il est possible de développer soi-même et assez facilement des plugins, que ce soit en Java™ ou en JavaScript².

1. Son code source est dans le domaine public et la plupart des plugins sont sous licence GNU-GPL
2. JavaScript est un langage de programmation de script orienté objet à prototype [4]

La figure ci-après donne un aperçu visuel de ce que nous voudrions obtenir au final. Nous pouvons y retrouver l'interface, l'image étudiée avec les particules sélectionnées, la pile d'images individuelles ainsi que le tableau de résultats.

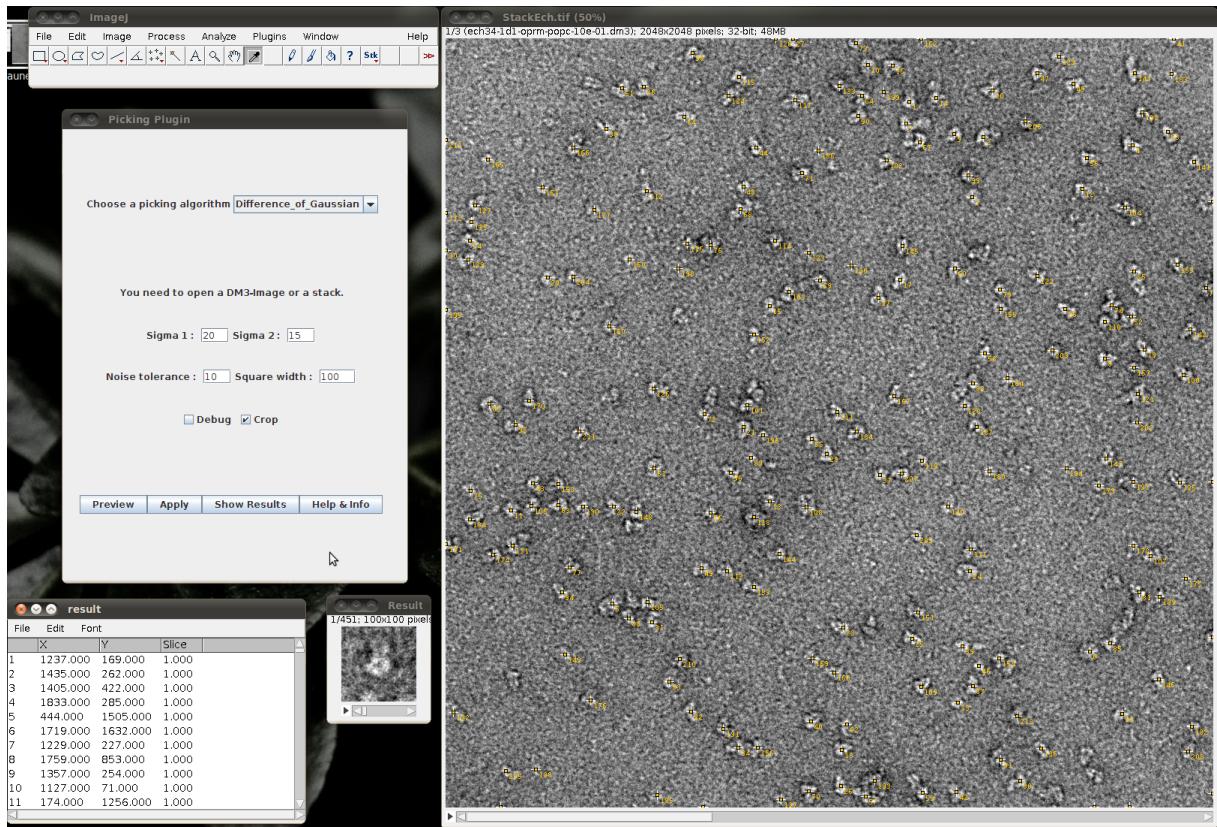


FIGURE 1.1 – Rendu visuel attendu du plugin de picking

1.2 État de l'existant

1.2.1 Détection à l'aide de références

Cette technique est utilisée pour trouver des particules dans une image en la comparant à un modèle. Celui-ci peut être obtenu soit à partir d'une structure tridimensionnelle connue, soit par sélection d'une particule servant d'exemple dans les micrographies. L'algorithme détermine la meilleure correspondance entre la cible et le modèle pour pouvoir la localiser dans l'image.

1.2.2 Piquage de particules sans référence

Détection des bords et transformée de Hough

Les difficultés majeures rencontrées dans les techniques basées sur la détection des contours sont dues à la complexité de détecter les bords des particules lorsqu'il y a un bruit de fond important sur les images de MET.

Cette technique est basée sur la détection des contours ainsi que sur l'application de la transformée de Hough [5]. Cette méthode permet de détecter la présence de formes comme des lignes, des cercles ou encore des ellipses.

Dans la transformée de Hough [6] appliquée à la détection de lignes, pour chaque pixel allumé de l'image on trace toutes les lignes possibles, on obtient alors une sinusoïde unique appelée espace de Hough. Si les courbes associées à deux points se coupent, l'endroit où elles se coupent dans l'espace de Hough correspond aux paramètres d'une droite qui relie ces deux points (ordonnée à l'origine et pente).

La détection de formes qui nous intéressent s'apparentant à des cercles ou des axes circulaires se fait en détectant le centre de celles-ci. Pour chaque pixel allumé, la fonction trace un cercle de rayon donné. Si la forme a le même rayon que le cercle tracé, tous les cercles se recoupent au centre de la forme, on constate ainsi une amplification de la valeur du pixel central.

D'autre part, pour détecter des particules de formes irrégulières, l'approche décrite précédemment peut également être utilisée mais la transformée de Hough devra alors être remplacée par la transformée de Hough Généralisée [7]. Cette nouvelle méthode repose sur la modification de la transformée de Hough en utilisant le principe de l'identification à partir d'un modèle de référence.

DoGLFC et classement par affinité

Cette technique est basée sur l'utilisation de l'algorithme DoGLFC³ complété par l'algorithme de classement par affinité [8].

Le DoGLFC est basé sur l'algorithme DoG Picker du Scripps Institute [9], une méthode rapide qui permet la segmentation de particules. Après l'application de l'algorithme de Différence de Gaussiennes (DoG), on obtient une cartographie de points similaire à celle de la méthode utilisant un modèle de référence.

Cet algorithme requiert un paramètre ajustable unique ou un jeu de paramètres basé sur le rayon de la particule ou un ordre de grandeur du rayon. L'exécution de cet algorithme renvoie une liste de trois paramètres décrivant la localisation de la particule (les coordonnées x et y) et la hauteur du pic.

L'algorithme DoGLFC sélectionne les particules (ou objets) potentielles de taille déterminée, ceci a un pouvoir discriminatif moindre par rapport à une technique basée sur un modèle de référence.

Pour améliorer le rendement lors du piquage des particules, un nouvel algorithme semi-supervisé, le classement par affinité, peut être appliqué.

L'algorithme a besoin de trois paramètres d'entrée : un jeu d'images, la taille minimale de l'image après réduction et deux références indiquant quelle fenêtre doit être utilisée comme référence positive ou négative.

Lorsque l'algorithme a fini de se dérouler, on obtient une classement pour chaque fenêtre où le maximum correspond à la particule ciblée.

L'utilisation de DoGLFC complété par le classement par affinité permet d'extraire rapidement les particules de l'image et d'éliminer avec précision les particules correspondant à de la contamination ou du bruit de fond.

3. Difference Of Gaussian Local Fast Correlation

1.2.3 Perceptron

Un perceptron est une sorte de réseau neuronal artificiel. Dans son état le plus simple, il représente un système de classification binaire/linéaire.

Ce programme a la particularité d'être capable d'apprendre des concepts, ce qui signifie qu'il peut apprendre afin de répondre par vrai ou faux à des données qui lui sont soumises, grâce à la présentation répétée de plusieurs exemples d'étude. Il a déjà été testé sur des images binaires pour la détection de formes ou de contours mais pas sur des images en niveaux de gris ou sur des problèmes de reconnaissance de modèles visant à sélectionner des particules. Le réseau neuronal n'a pas non plus été exploité comme un outil de sélection automatique de particules mais plusieurs recherches ont conclu qu'il pourrait être utilisé pour l'élimination de faux-positifs. [10].

1.3 Analyse des besoins

Ils sont de deux types : fonctionnels et non fonctionnels, et diffèrent entre l'interface utilisateur et les algorithmes de piquage.

1.3.1 Besoins fonctionnels

Interface

Au lancement, les images sont préalablement chargées sur ImageJ, l'utilisateur a le choix entre plusieurs algorithmes de piquage. Nous avons essayé de faire en sorte que l'affichage soit clair et succinct, constitué d'un menu déroulant, d'une liste de boutons et d'une interface propre à chaque algorithme.

Enfin, il est également possible d'implanter simplement de nouveaux algorithmes dans l'interface.

Les algorithmes

Les algorithmes implantés doivent réaliser un piquage automatique des particules depuis les micrographies issues de MET. Le format de sortie est un tableau de résultats contenant un jeu de coordonnées x, y associé à la position de l'image dans la pile d'images (*stack* en anglais), ainsi qu'une nouvelle pile contenant les particules sélectionnées par l'algorithme si l'utilisateur le désire.

1.3.2 Besoins non fonctionnels

Interface

L'implémentation de la plateforme a été réalisée en Java TM, nous utilisons les API⁴ graphiques de la bibliothèque *Swing*. Il s'agit d'une interface graphique (GUI)⁵ faisant partie du paquetage Java Foundation Classes (JFC), inclus dans J2SE. Cela constitue une des principales évolutions apportées par Java2 par rapport aux versions antérieures. Elle offre la possibilité de créer des interfaces graphiques identiques quelque soit de système d'exploitation sous-jacent, au prix de performances moindres qu'en utilisant Abstract Window Toolkit (AWT) l'autre bibliothèque principale pour Java TM.

4. Application Programming Interface

5. Graphic User Interface

De plus, nous avons essayé de faire en sorte que le programme soit le plus simple d'utilisation possible afin de le rendre accessible à tous. C'est pourquoi nous avons tenté de faire un code clair, explicite et commenté afin de permettre aisément l'implémentation de nouveaux algorithmes.

Les algorithmes

Les algorithmes ont été testés avec l'outil macro d' ImageJ puis implémentés en Java TM. Nous avons fait en sorte que le temps de déroulement de l'algorithme soit assez rapide afin de pouvoir gérer de grands jeux de données et qu'aucune image de traitement intermédiaire n'apparaisse à l'utilisateur (à moins qu'il ne le souhaite).

Dans l'optique de démocratiser l'utilisation de notre interface et afin de permettre l'extension de ce plugin avec d'autres algorithmes, le projet a été placé sous une licence GPL⁶.

1.4 Scénario d'utilisation du plugin

Nous prendrons l'exemple d'une image issue de cryo-MET collectée dans l'équipe ACMPC sur laquelle se trouvent des protéines membranaires que nous voulons sélectionner (Figure 1.3) ainsi que d'une image utilisée comme référence sur ImageJ , les *blobs* (Figure 1.2).

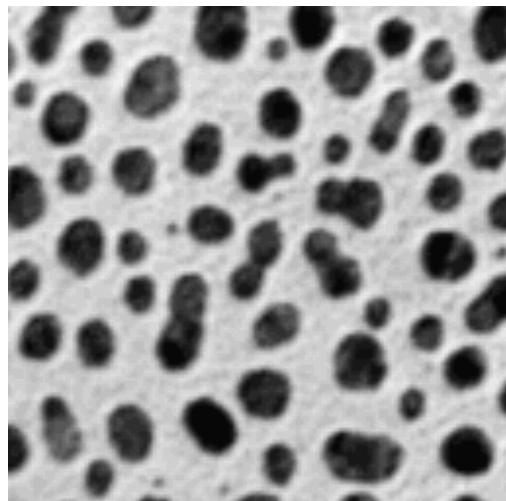


FIGURE 1.2 – Image Blobs de référence

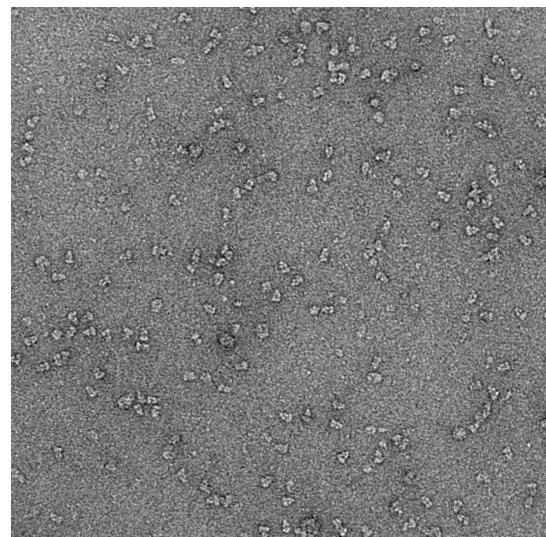


FIGURE 1.3 – Image de protéines membranaires observées au MET

Nous avons tenté de nous mettre à la place d'un chercheur pour avoir une vision biologique de l'utilisation de notre plugin. Tout au long de ce rapport, vous pourrez donc suivre toutes les procédures qui nous ont permis de réaliser la sélection.

6. General Public License [11]

2.1 Interface graphique

Comme précisé précédemment, notre programme se présente sous la forme d'un plugin ImageJ. L'interface graphique a été réalisée grâce à la bibliothèque Java™ Swing. La fenêtre servant d'interface est composée de différentes boîtes (panneaux) contenant différents outils (boutons, listes, zones de saisie et zones de texte) permettant l'interaction entre l'utilisateur et le programme. Ces boîtes peuvent s'imbriquer les unes dans les autres afin d'obtenir des structures plus complexes et un meilleur rendu visuel.

L'interface du plugin est simple, elle prend la forme d'une fenêtre composée d'un menu déroulant, dans lequel se trouvent les différents algorithmes, et de quatre boutons.

2.1.1 Panneaux

La Figure 2.1 donne un aperçu de l'organisation des différents panneaux de notre interface. Celle-ci est composée d'une arborescence de panneaux comportant un panneau principal (en bleu, permettant d'organiser la fenêtre du plugin), lui-même constitué des trois panneaux subsidiaires (en orange) suivants :

Le premier panneau contient une liste déroulante permettant le choix de l'algorithme.

Le panneau du milieu est vide au lancement du plugin et s'actualise en fonction de l'algorithme de piquage choisi. Il affiche une aide rapide sur les conditions d'utilisation de l'algorithme de piquage, les zones de saisies utilisateur propre à cet algorithme. Il affiche également les choix génériques : mode de débogage, mode de découpe des sélections et taille de la sélection, gestion du bruit de fond.

Tout cela s'organise à l'aide de sous-panneaux (en violet).

Le dernier panneau contient les boutons d'actions : prévisualisation, exécution, affichage des résultats, informations/aide.

Le panneau principal permet notamment d'imposer la taille de l'interface.

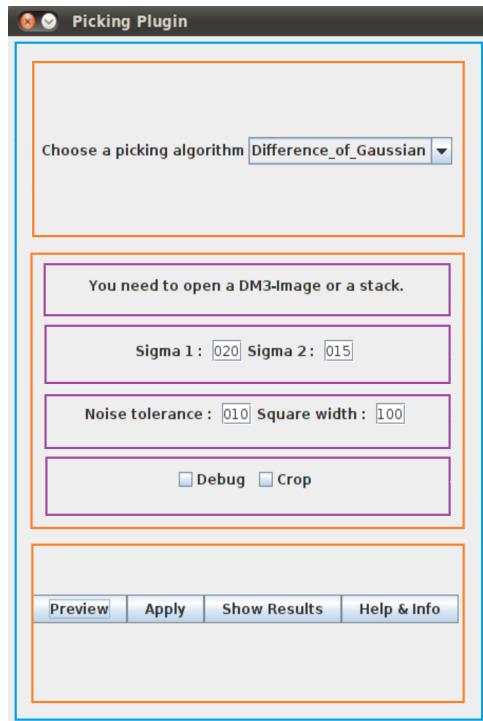


FIGURE 2.1 – Organisation des panneaux pour l'algorithme Difference of Gaussian

2.1.2 Boutons

Les boutons doivent répondre aux clics de la souris et lancer l'action correspondante en fonction de l'algorithme de piquage choisi par l'utilisateur :

Le bouton de prévisualisation permet de tester le piquage obtenu par l'algorithme sélectionné sur l'image courante.

Le bouton d'exécution permet d'appliquer l'algorithme sur l'image ou la pile d'images sélectionnée, lorsque l'utilisateur est sûr des paramètres qu'il souhaite appliquer.

Le bouton d'information/aide est une aide d'utilisation du plugin ainsi que des conditions de fonctionnement des algorithmes.

Le bouton d'affichage des résultats permet d'afficher et de sauvegarder le tableau des coordonnées des particules sélectionnées.

Pour que le plugin puisse fonctionner, l'utilisateur devra au préalable avoir ouvert une image ou un *stack* d'images à l'aide d'ImageJ.

2.2 Récupération des paramètres

Les méthodes de piquage ont une classe mère commune. L'utilisateur sélectionne un algorithme dans le menu déroulant et détermine certains paramètres. Quelques uns sont propres aux algorithmes et d'autres sont communs à tous tels que :

- la gestion du bruit de fond,
- la taille des images des particules sélectionnées lorsqu'on demande le *stack* d'images,
- l'activation du débogage (s'il est implémenté dans l'algorithme),
- l'activation du découpage des particules à partir du piquage obtenu.

2.3 Algorithmes

2.3.1 Corrélation d'image

La méthode par corrélation croisée donne des résultats similaires à une transformée de Hough pour des objets circulaires. Elle produit une carte de corrélation entre une référence (dans notre cas, un cercle de taille définie) et une image de MET contenant des particules circulaires. Chaque pixel de cette carte correspond à la mesure de similarité entre les deux images. Les valeurs élevées des pixels représentent donc les meilleures similarités et permettent de localiser les particules circulaires de même rayon que celle de l'image de référence. Cette opération est réalisée en espace fréquentiel via une Transformée de Fourier et est déjà implantée dans ImageJ.

Par rapport à la transformée de Hough, la corrélation croisée est beaucoup plus rapide mais est réservée (pour l'implantation dans ImageJ) à des images carrées dont le côté doit être une puissance de 2. Cet inconvénient ne nous gêne pas car les images de MET sont de 2048x2048 pixels (et la plupart des traitements sont appliqués à des images réduites de 1024x1024).

Pour cette méthode de piquage, plusieurs images de référence sont utilisées avec des cercles de divers rayons afin de sélectionner toutes les particules circulaires des images de MET. L'utilisateur doit entrer les rayons minimal et maximal des particules à sélectionner, ainsi que la valeur de l'incrément.

Pour un résultat optimal, avant de lancer la sélection des particules avec cet algorithme, l'utilisateur devra traiter l'image pour éliminer un maximum de bruit de fond (utilisation de filtres).

2.3.2 Différence de Gaussiennes (DoG pour *Difference of Gaussian*)

La différence de Gaussiennes est une technique qui soustrait une image à laquelle a été appliqué un filtre gaussien à une deuxième image, elle aussi filtrée (par le même filtre) mais en utilisant une valeur d'écart-type plus petite, donc moins filtrée. Par conséquent, les bords des particules sont dégradés, nous permettant de récupérer les maxima de l'image résultante, c'est-à-dire le centre des particules.

Il est demandé à l'utilisateur d'entrer les valeurs d'écart-type qui seront utilisées pour appliquer les filtres gaussiens.

2.3.3 Extraction de contours

Cette méthode (aussi appelée "Différence de dilatation") repose sur le même principe que la Différence de Gaussiennes, mais en utilisant des opérateurs de morphologie mathématique (érosion et dilatation). Nous lui appliquons un certain nombre de cycles de dilatation et obtenons alors des particules de plus grande taille. A la soustraction des deux images, seuls les contours des particules apparaîtront, nous permettant de récupérer le centre de particules.

L'utilisateur devra entrer le nombre de cycles de dilatation qu'il souhaite appliquer à chaque image.

2.4 Pile d'images résultantes

Si l'utilisateur le désire, il peut récupérer une pile d'images contenant les particules piquées. Celle-ci est obtenue à partir du tableau des coordonnées que la fonction de création du *stack* prend en paramètre d'entrée. Les particules trop près du bord sont éliminées, les autres ajoutées dans un *stack* qui est finalement affiché à l'utilisateur.

2.5 Organisation orientée objet du programme

Afin de clarifier et de simplifier le code, nous avons tenté de séparer les grandes fonctions du programme en suivant les principes de la programmation orientée objet.

2.5.1 Séparation interface graphique et traitement d'images

Sans suivre le patron de conception Modèle-Vue-Contrôleur (MVC [12]), nous avons créé différentes classes permettant de distinguer la partie GUI de la partie algorithme. Nous obtenons ainsi des classes ne gérant que l'aspect graphique du plugin. Celles-ci gèrent plus particulièrement les panneaux propres aux méthodes de piquage ainsi que les classes bien distinctes effectuant les traitements d'images.

2.5.2 Séparation traitement d'images et création du *stack*

La partie permettant la création du *stack* d'images individuelles a été séparée des méthodes de piquage afin d'éviter la répétition de ce code dans chaque algorithme. Cela permet aussi d'y faire appel en dehors de nos algorithmes, contrairement à une implémentation de cette fonction dans une classe mère.

2.5.3 Patrons de conception

Cette séparation GUI/traitement/*stack* à dû être accompagnée de la création de classes suivant des patrons de conception particulier : la *factory* et le *singleton*.

Le ***singleton*** est un patron de conception (*design pattern*) permettant de restreindre l'instanciation d'une classe à un seul objet (ou bien à quelques objets seulement).

La classe **Attributes** en *singleton* permet de conserver les paramètres choisis par l'utilisateur. De ce fait, même lorsqu'il exécute plusieurs algorithmes, une seule instance de cette classe peut exister, actualisant les paramètres choisis par l'utilisateur.

La **factory** définit une interface pour la création d'objets (ou groupes d'objets). La classe **AlgoFactory** en *factory* rend possible les transitions entre chaque méthode de piquage. Elle amorce le panneau propre à l'algorithme permettant d'actualiser l'interface et ainsi de récupérer les paramètres utilisateur. Ensuite, lorsque l'utilisateur clique sur les boutons d'exécution ou de prévisualisation, la *factory* lancera les actions correspondantes. Afin d'éviter des erreurs d'instanciations, cette classe suit également le patron de conception *singleton*.

2.5.4 Modularité et réutilisation du code

L'organisation orientée objet du programme a fortement facilité l'écriture du programme, évitant la répétition de lignes de code qui pouvaient être factorisées. De plus, cela permet une grande modularité au sein de notre programme, facilitant l'ajout ou le retrait d'un algorithme de piquage et de son interface dans notre plugin. Cette modularité est telle qu'elle permet, en ce qui concerne le module de création du *stack* de particules sélectionnées, de l'utiliser en dehors de notre plugin.

Cela tient un rôle important dans notre projet car, comme pour beaucoup de logiciels libres, ImageJ se développe énormément grâce à sa communauté d'utilisateurs qui participe au développement de plugins. Après validation par le personnel du NIH¹ responsable de ce projet, les plugins sont mis à la disposition des utilisateurs. Ainsi, chacun pourrait ajouter/améliorer des méthodes de piquage de particules, développant et augmentant l'efficacité et la sélectivité de notre plugin.

Ci-après (Figure 2.2) le diagramme général de l'organisation de nos classes :

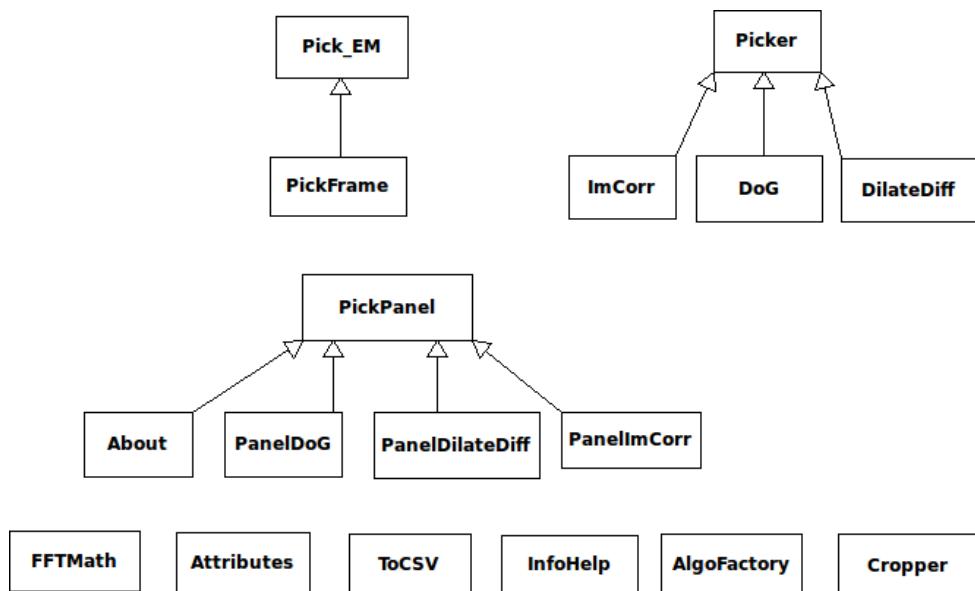


FIGURE 2.2 – Organisation générale des classes du plugin Pick_EM

1. National Institute of Health

Afin de faciliter la portabilité de notre plugin, nous avons choisi de le compresser au format *JAR*² qui est un fichier ZIP³ utilisé pour distribuer un ensemble de classes Java™. Ce format est utilisé pour stocker les définitions des classes, constituant l'ensemble d'un programme. Ce format est directement pris en compte par ImageJ et, lorsque l'archive est placée dans le dossier correspondant, directement ajouté au menu des plugins.

Cette archive contient les fichiers source *.java* pour permettre la modification du code et les fichiers compilés *.class* afin de permettre l'utilisation du plugin sans avoir de problèmes de compilation (généralement dus à la version de Java™).

Du fait que notre plugin soit *open-source* et en libre accès, il était important de protéger l'accès à notre travail. Pour cela, nous avons décidé de placer notre programme sous licence.

Plusieurs types de licence étaient envisageables, notamment les licences *GPL*, *L-GPL*, *BSD* pour celles en anglais, et les licences *CéCill* pour celles propres à la législation française. Sur l'avis de nos tuteurs, mais aussi parce que c'est celle la plus couramment utilisée pour les plugins ImageJ, nous avons choisi la licence *GPL* qui est la plus permissive tout en nous garantissant l'accès à notre travail.

2. Java ARchive [13]
3. Format de fichier permettant l'archivage et la compression de données sans perte de qualité [14]

Réalisation

Sous ImageJ, il existe deux types de plugins : *PlugInFilter* et *PlugInFrame*. Étant donné que nous avions besoin de différents objets graphiques, nous avons choisi d'utiliser *PlugInFrame*. C'est la classe `Pick_EM` qui en hérite et permet de lancer notre plugin à son appel par l'intermédiaire de la barre de menus d'ImageJ.

3.1 Interface graphique

3.1.1 Panneaux et boutons

Nos panneaux dérivent de la classe *JPanel*, elle-même issue de la classe *Panel*. Cette dernière fournit un composant *Container* permettant d'accueillir d'autres composants graphiques (sous-panneaux).

Le premier panneau (**panel1**) contient une zone de texte (*JLabel*) afin d'afficher un message d'aide, ainsi qu'un menu déroulant (*JComboBox*) pour le choix des algorithmes.

Le panneau central (**panel2**) est vide au lancement du plugin et son contenu varie en fonction de l'algorithme sélectionné. Par exemple pour l'algorithme Difference of Gaussian, les sous-panneaux sont créés dans la classe `panelDoG`. Pour cet algorithme, le **panel2** comprend :

- **infoPanel** contenant un *JLabel* indiquant le type d'image requis.
- **sigmaPanel** et **widthNoisePanel** qui contiennent des *JLabel* et *JTextField* afin de créer une zone dans laquelle il est possible d'entrer les paramètres nécessaires au déroulement du piquage.
- **debugCropPanel**, quant à lui, contient deux cases à cocher (*JCheckBox*) pour activer ou non les modes de débogage et de crop.

Les classes `panelImCorr` et `panelDilateDiff` servent à la création des sous panneaux des algorithmes Image Correlation et Dilate Difference respectivement.

Le dernier panneau (**panel3**) comporte quatre boutons (*JButton*) devant répondre aux clics de la souris à l'aide d'un *ActionListener*.

Enfin, le panneau principal (**mainPanel**) contient tous les panneaux cités précédemment. Sa taille détermine celle de la fenêtre du plugin.

Ces quatre panneaux sont créés dans la classe `PickFrame`, qui hérite de la classe `JFrame`. `PickFrame` peut également accéder à des méthodes de la classe `ActionListener`.

La figure suivante (Figure 3.1) donne un aperçu plus visuel de l'organisation de ces différents panneaux.

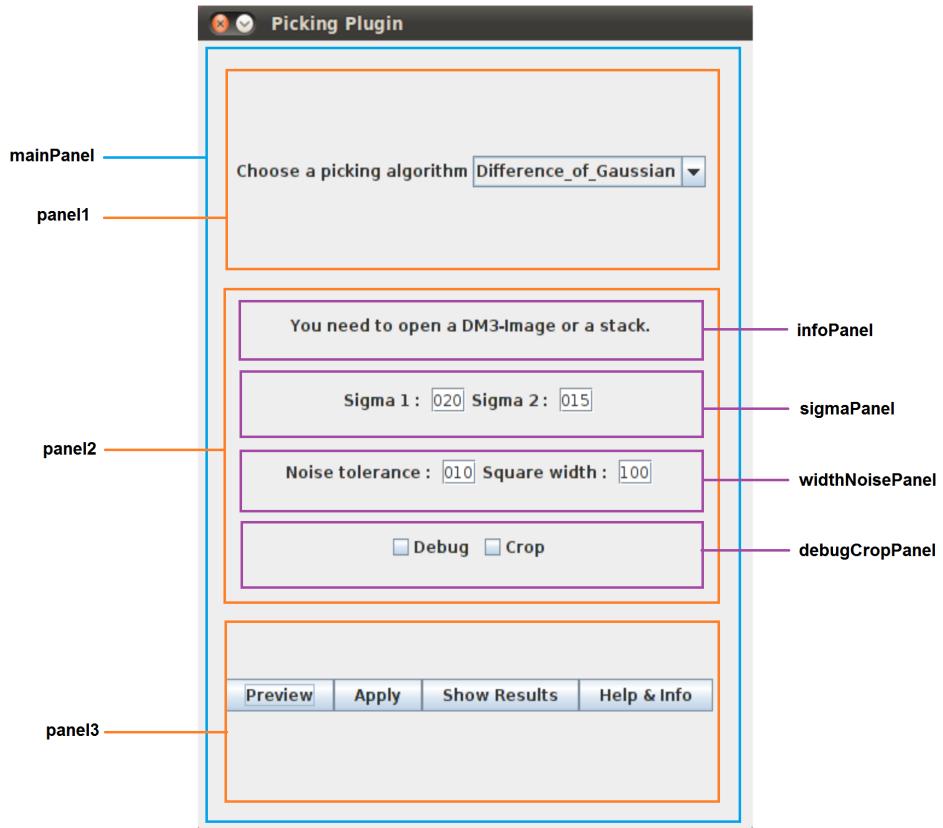


FIGURE 3.1 – Organisation des panels pour l'algorithme Difference of Gaussian

La classe `PickFrame` permet donc de générer l'interface graphique du plugin. De plus, elle permet de réagir lors d'un clic sur un bouton. C'est la méthode `actionPerformed()` qui vérifie le nom de l'algorithme choisi par l'utilisateur parmi ceux proposés dans la JComboBox et permet d'afficher le panel2 qu'il faut. Ci-après un extrait de la méthode `actionPerformed()` :

```

public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    String comboSelection = null;
    if (command.equals("comboBoxChanged")){
        JComboBox cb = (JComboBox)e.getSource();
        comboSelection = (String) cb.getSelectedItem();
        panel2.removeAll();
        // Allows the panel2's update
        mainPanel.remove(panel1);
        mainPanel.remove(panel2);
        mainPanel.remove(panel3);
        panel2 = AlgoFactory.algorithm.getPickPanel(comboSelection);
        mainPanel.add(panel1);
        mainPanel.add(panel2);
        mainPanel.add(panel3);
        mainPanel.repaint();
        pack();
    }
}

```

Comme nous avions quelques soucis d'actualisation des panneaux, lors de l'implémentation, nous avons décidé de les supprimer puis de les recréer lors de chaque changement d'algorithme. Le chargement du bon panel2 se fait grâce à `getPickPanel` de la classe `AlgoFactory`. La méthode `pack()` permet de recadrer la fenêtre en fonction de son contenu.

```

if (command.equals("Apply")){
    comboSelection = (String) algoList.getSelectedItem();
    Attributes.getInstance();
    coordXYZ = AlgoFactory.algorithm.getPicker(comboSelection);
    IJ.showStatus("End of picking");
}
else if (command.equals("Preview")){
    comboSelection = (String) algoList.getSelectedItem();
    Attributes.getInstance();
    AlgoFactory.algorithm.getPickerPreview(comboSelection);
    IJ.showStatus("End of Preview");
}
else if (command.equals("Show Results")){
    ToCSV.generateCsvFile(coordXYZ);
}

```

Grâce à cette série de structures conditionnelles, il est possible d'appliquer la méthode de piquage appropriée en fonction du choix de l'utilisateur. Le lancement de la procédure ne se fait que si ce dernier clique sur les boutons de prévisualisation ou d'application. La tableau de résultats ne sera généré que s'il appuie sur le bouton d'affichage des résultats.

3.1.2 Affichage des résultats

Lorsque l'utilisateur a coché la case "crop" avant de lancer la procédure de piquage, la classe **PickFrame** fait appel à la classe **Cropper** permettant de créer un *stack*. Les paramètres d'entrée de **Cropper()** sont une *ImagePlus* (image courante du *stack*) et un tableau de doubles contenant les coordonnées des particules sélectionnées.

Lors de nos phases de tests, nous avons ajouté une autre fonction **Cropper()**, sans paramètres d'entrée. Nous y avons créé un tableau de coordonnées manuellement pour faciliter les essais.

A partir du tableau de doubles cité, la méthode **crop()** de **Cropper** fait appel à la méthode **setRoi()** d'**ImageJ** afin de retenir une zone carrée autour de la sélection. Cette zone va ensuite être dupliquée et ajoutée au *stack* sous la forme d'un *ImageProcessor*. Ci-dessous un extrait de la méthode **crop()** :

```
if (z == currentSlice) {  
    imp.setRoi(x, y, widthCrop, widthCrop);  
    // widthCrop = taille du carre de selection entre par l'utilisateur  
    img2 = new Duplicator().run(imp);  
    ImageProcessor ip2 = img2.getProcessor();  
    ImageProcessor impTemp = ip2.resize(widthCrop, widthCrop);  
    ims.addSlice(impTemp);  
}  
}
```

Cette partie du code ne va s'exécuter que si le cadre de sélection de la particule ne dépasse pas le cadre de l'image de base. Les particules dépassant n'apparaitront pas dans le *stack* d'images individuelles, mais on pourra retrouver leurs coordonnées dans le tableau de résultats final.

Par ailleurs, les résultats de la sélection peuvent être affichés sous la forme d'une **ResultsTable** si on clique sur le bouton "Show Results". Elle est construite grâce au tableau de doubles cité précédemment, qui est le paramètre d'entrée de la fonction **generate-CsvFile()**.

3.2 Récupération des paramètres

Le singleton de la classe **Attributes** contient une table de hashage (*HashTable*) dans laquelle sont stockés tous les paramètres entrés par l'utilisateur. Ces derniers sont accessibles grâce à des clés et donc réutilisables dans les algorithmes. La méthode **synchronized()** dans la fonction **getInstance()** (voir ci-dessous) empêche toute instanciation multiple.

```
public final static Attributes getInstance() {  
    if (Attributes.instance == null) {  
        synchronized(Attributes.class) {  
            if (Attributes.instance == null) {  
                Attributes.instance = new Attributes();  
            }  
        }  
    }  
    return Attributes.instance;  
}
```

3.3 Algorithmes

3.3.1 Comparaison langage Macro ImageJ et Java™

Nous avons commencé par implémenter les algorithmes grâce à l'outil Macro d'ImageJ lors de nos phases de test. Nous les avons ensuite traduits en Java™ et liés à la partie interface graphique du code. Vous trouverez ci-après un exemple de cette transformation.

Extrait de la Macro de l'Extraction de contours (Dilate Difference) :

```
run("Blobs (25K)");  
run("Duplicate...", "title=blobs-1.gif");  
run("Duplicate...", "title=blobs-2.gif");  
run("Dilate");  
run("Make Binary");  
selectWindow("blobs-1.gif");  
run("Make Binary");  
run("Dilate");  
run("Options...", "iterations=2 count=1 edm=Overwrite do=Nothing");  
selectWindow("blobs-2.gif");  
run("Dilate");  
imageCalculator("Subtract create", "blobs-2.gif", "blobs-1.gif");  
selectWindow("Result of blobs-2.gif");  
run("Find Maxima...", "noise=3 output=[Point Selection]");
```

Équivalent en langage Java™ :

```
ImagePlus imp = WindowManager.getCurrentImage();
ImagePlus imp1=new Duplicator().run(imp);
ImagePlus imp2= new Duplicator().run(imp1);
imp1.setSlice(currentslice);
imp2.setSlice(currentslice);
IJ.run(imp1, "Make Binary", "calculate");
IJ.run(imp2, "Make Binary", "calculate");
IJ.run(imp1, "Options...", it1);
IJ.run(imp1, "Dilate", "slice");
IJ.run(imp2, "Options...", it2);
IJ.run(imp2, "Dilate", "slice");
ic = new ImageCalculator();
ImagePlus imp3 = ic.run("Subtract create", imp2, imp1);
ImageProcessor ip3 = imp3.getProcessor();
ip3.invert();
Polygon points = mf.getMaxima(ip3, tolerance, excludeOnEdges);
```

3.3.2 Méthodes de piquage

Lorsque l'utilisateur fait le choix d'un algorithme de piquage parmi ceux qui lui sont proposés, cela fait appel à la classe **AlgoFactory** contenant plusieurs méthodes *switch* :

- **getPickPanel()** permet de récupérer le nom de l'algorithme choisi et d'afficher le panel2 correspondant.
- **getPicker()** permet de lancer le piquage lorsque l'utilisateur appuie sur le bouton Apply.
- **getPickerPreview()** permet de lancer le piquage lorsque l'utilisateur appuie sur le bouton Preview.

La présence d'un constructeur privé dans cette classe supprime le constructeur public par défaut. De plus, seul le singleton peut s'instancier lui-même.

Une fois le panel2 chargé, l'appel aux procédures de piquage ne peut se faire que si l'on clique sur les boutons de prévisualisation (Preview) ou d'application à l'ensemble du *stack* (Apply).

Les paramètres entrés par l'utilisateur sont sauvegardés dans la table de hashage grâce à la fonction **getInstance()** de la classe **Attributes**. Dans le cas où l'utilisateur entrerait des valeurs non numériques, c'est ImageJ qui se chargerait de gérer les erreurs.

Les paramètres sont ensuite récupérés, pour l'algorithme, par la fonction **setAttributes()** de la classe **PanelDoG** (pour suivre notre scénario).

L'algorithme est par la suite appelé par la méthode **sliceSelection()** (Apply) ou par **picking()** (Preview).

La méthode **picking()** donne l'image courante du *stack* en paramètre de la méthode **pick()** alors que **sliceSelection()** parcourt le *stack* et appelle **pick()** autant de fois qu'il y a d'images dans le *stack*.

La méthode **pick()**, quant à elle, permet de lancer l'algorithme sur la sélection. Elle prend une **ImagePlus** et le numéro de l'image dans le *stack* en paramètres. Cette fonction

récupère les paramètres entrés par l'utilisateur (grâce à `hashAttributes.get()`) et renvoie un tableau de résultats (X, Y, Slice). Notons que slice représente une image dans la pile. Voici un extrait du code de l'algorithme Difference of Gaussian :

```
Hashtable<String , String> hashAttributes = Attributes .getAttributes ();  
String sigma1 = hashAttributes .get ("sig1 ");  
  
imp .setSlice (currentslice );  
ImagePlus imp1 = new Duplicator () .run (imp );  
  
String si1 = "sigma=" + sigma1 ;  
  
imp1 .setSlice (currentslice );  
IJ .run (imp1 , "Gaussian Blur ..." , si1 );
```

Ici, nous obtenons le paramètre `sigma1` grâce à la table de hashage et à la clé "`sig1`". Nous avons besoin de ce dernier pour appliquer le filtre gaussien sur l'image courante, c'est pourquoi nous le convertissons sous la forme d'un *String*. Notons que la fonction `IJ.run()` permet de lancer une procédure ImageJ.

Nous avons choisi d'utiliser des vecteurs pour stocker les coordonnées ainsi que les numéros de *slices* car il nous est impossible de connaître à l'avance le nombre de particules qui vont être sélectionnées.

Nous avons fait en sorte de vider les tableaux de résultats entre le mode de prévisualisation et d'application, mais aussi entre deux applications ou deux prévisualisations. Ceci évite que les résultats ne s'ajoutent, ce qui fausserait la résultante du piquage. Il en est de même pour tous les algorithmes.

De plus, les classes contenant les algorithmes de piquage héritent de la classe `Picker`. Celle-ci contient la méthode `resultConverter()` permettant regrouper les différentes coordonnées ainsi que les numéros des *slices* dans un seul et même tableau. De plus, tous les attributs communs aux algorithmes sont déclarés dans cette classe.

3.4 Autres classes

La classe `FFTMath` est issue de la classe `FFTMath` d'ImageJ, que nous avons modifié afin de pouvoir réaliser la corrélation d'images dans l'algorithme Image Correlation. Nous avons choisi de le modifier afin d'éviter un affichage graphique intempestif, qui avait pour effet de ralentir le déroulement du programme. De plus, cela n'avait pas d'intérêt particulier pour l'utilisateur.

La classe `About` permet d'afficher des informations, ainsi que le moyen de nous contacter en tant qu'auteurs du plugin si besoin est.

La classe `InfoHelp` affiche une aide sur le fonctionnement du plugin si l'utilisateur clique sur le bouton "Help & Info".

3.5 Applications

Les sélections de piquage affichées sur l'image correspondent aux positions des particules sur l'image courante, ou sur la dernière image dans le cas d'un *stack*.

L'utilisateur peut choisir d'afficher un tableau contenant les coordonnées (abscisses, ordonnées et positions dans le *stack*) des particules sélectionnées et pourra le sauvegarder.

De plus, s'il le désire, un *stack* contenant les particules sélectionnées aux positions obtenues est créé (éliminant les particules trop près du bord de l'image) et affiché.

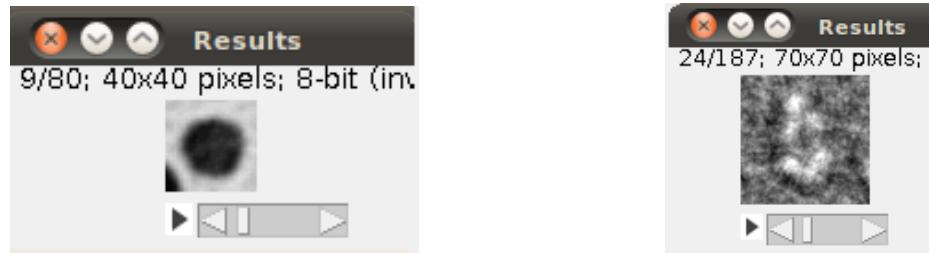


FIGURE 3.2 – Exemple d'images formant le *stack* de particules (blobs et protéines)

Statistiques

- La **Différence de Gaussiennes** permet d'obtenir les résultats suivants :

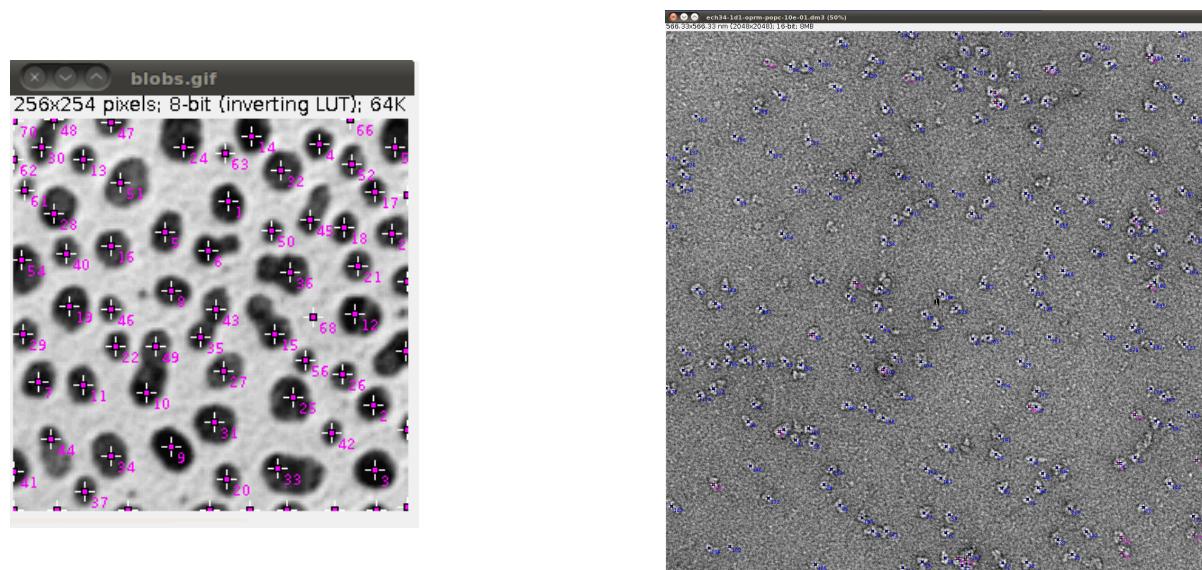


FIGURE 3.3 – Différence de Gaussiennes (blobs et protéines)

Nous constatons que ce piquage est efficace sur notre image de protéines membranaires. Nous l'avons également testé sur une image de blobs, qui est une image de référence pour les essais sur ImageJ, elle semble tout aussi bien marcher.

- L'**Extraction de contours** permet d'obtenir les résultats suivants :

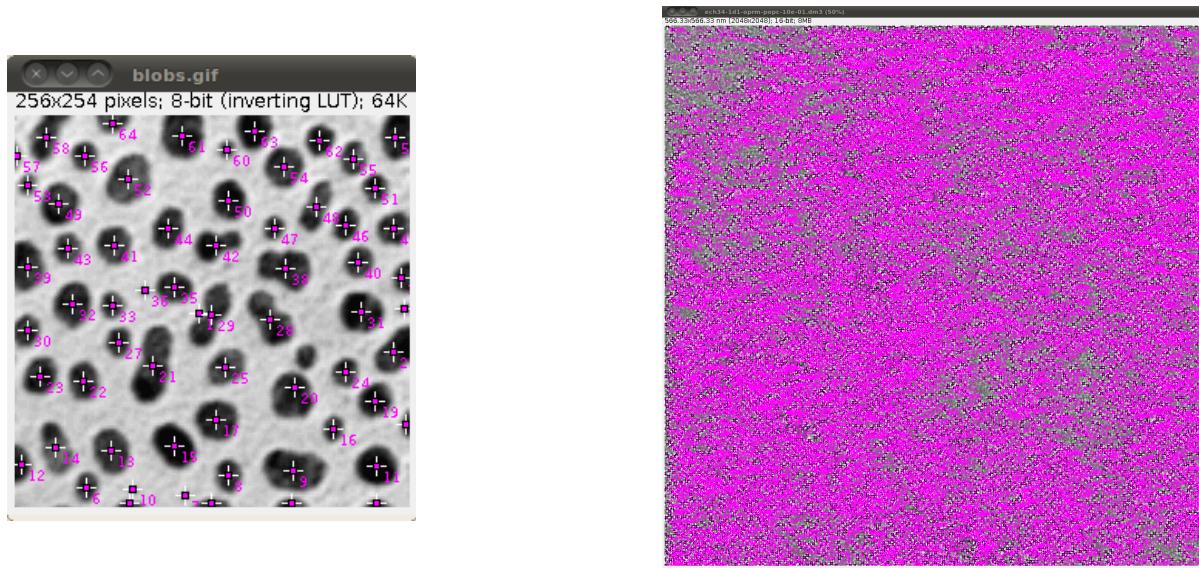


FIGURE 3.4 – Extraction de contours (blobs et protéines)

Nous observons que cet algorithme semble bien fonctionner sur les blobs, alors que sur les protéines il y a beaucoup trop de points de piquage. Même en modifiant le paramètre de la "Noise Tolerance" nous obtenons des résultats de piquage similaires.

- La **Corrélation d'images** ne permet pas d'obtenir de résultats de piquage exploitables avec ces deux types d'images. Nous avons également testé cet algorithme avec des images de particules virales. Nous obtenions de bon résultats, cependant, pour des raisons de confidentialité, nous ne pouvons pas intégrer ces données dans notre rapport.

- **Résultats :**

Nous avons réalisé une série de tests statistiques afin de comparer l'efficacité de nos algorithmes sur les différentes images. Les résultats obtenus sont regroupés dans le tableau (Table 3.1) suivant :

Images	Variables	DoG	Dilate Difference	Image Correlation
Blobs	Vrai Positif (VP)	63	61	null
	Vrai Négatif (VN)	3	0	null
	Faux Positif (FP)	9	2	null
	Faux Négatif (FN)	0	1	null
	Sensibilité (SE)	1	0.98	null
	Spécificité (SP)	0.25	0	null
Protéines	Vrai Positif (VP)	167	infini	null
	Vrai Négatif (VN)	8	infini	null
	Faux Positif (FP)	16	infini	null
	Faux Négatif (FN)	8	infini	null
	Sensibilité (SE)	0.95	null	null
	Spécificité (SP)	0.5	null	null

TABLE 3.1 – Tableau de statistiques d'efficacité des algorithmes de piquage

Les **Vrais Positifs** sont les particules devant être sélectionnées et qui le sont par l'algorithme. Les **Vrais Négatifs** représentent tout ce qui ne doit pas être sélectionné et qui ne l'est pas. Les **Faux Positifs** correspondent à tout ce qui ne doit pas être sélectionné mais qui l'est. Enfin, les **Faux Négatifs** sont les particules devant être sélectionnées mais qui ne le sont pas.

La **Sensibilité** [15] est la probabilité qu'une particule devant être piquée le soit. Une mesure de la sensibilité s'accompagne toujours d'une mesure de la spécificité. La **Spécificité** est la probabilité de ne pas sélectionner ce qui ne doit pas l'être. La sensibilité et la spécificité sont obtenues par les formules suivantes :

$$SE = \frac{VP}{VP+FN} \text{ et } SP = \frac{VN}{VN+FP}$$

D'après la Table 3.1, nous remarquons que c'est l'algorithme de Différence de Gaussiennes qui a la meilleure spécificité et sensibilité pour les blobs. En effet, il y a plus de particules sélectionnées avec ce dernier qu'avec l'Extraction de contours, alors que cet algorithme a été créé pour les blobs au départ. Cependant, nous constatons qu'il y a plus de faux positifs avec la Différence de Gaussiennes.

En ce qui concerne les protéines, l'Extraction de contours est inefficace car elle ne fait pas de distinction entre le bruit et les particules. La Différence de Gaussiennes, quant à elle, est assez efficace en ce qui concerne la sélection de ces dernières.

3.6 Ajout d'un nouvel algorithme au plugin

Voici la marche à suivre pour ajouter un nouvel algorithme de piquage à notre plugin :

- Ajouter le nom de l'algorithme dans la liste des noms proposés par la *JComboBox* dans la classe *PickFrame*.
- Ajouter un nouveau "case" dans chaque "switch" de la classe *AlgoFactory*.
- Créer une nouvelle classe *PanelAlgorithme* dans laquelle doivent se trouver :
 - un *JLabel* pour la phrase d'indication d'utilisation de l'algorithme,
 - tous les *JLabel*, *JCheckBox*, *JButton*, *JTextField*, *JPanel* nécessaires pour le fonctionnement de l'algorithme,

- une méthode `setAttributes()` pour récupérer les paramètres entrés par l'utilisateur,
- Créer une nouvelle classe `Algorithme` pour la procédure de piquage suivant le modèle de ceux déjà implantés.

3.7 Améliorations possibles

Nous avons fait en sorte que notre plugin corresponde le plus possible aux attentes de notre tuteur. Cependant, quelques améliorations restent possibles.

Une fonction pour ajouter automatiquement un nouvel algorithme de piquage avec ses paramètres d'entrée via l'interface, sans que l'utilisateur n'ai à toucher au code source, serait pratique. Dans ce cas, il n'y aurait plus qu'à ajouter une classe implémentant cette nouvelle méthode de piquage dans le code.

Le mode de création du *stack* d'images individuelles pourrait éviter d'afficher les images dupliquées au cours de la création du *stack*. Cela permettrait d'alléger la mémoire vive de l'ordinateur, ainsi que de diminuer le temps de traitement.

De plus, il n'est actuellement pas possible de visualiser les sélections sur toutes les images du *stack* de base en même temps. Il serait donc intéressant de pouvoir le faire.

Par ailleurs, le *stack* d'images individuelles peut contenir des éléments qui n'auraient pas dû être sélectionnés ("paquets", bruit de fond). Il serait utile de pouvoir les supprimer du *stack* ou d'ajouter l'image d'une particule n'ayant pas été sélectionnée automatiquement.

La gestion de la mémoire pourrait être améliorée en optimisant nos algorithmes (par exemple, en diminuant le nombre d'objets créés lors de l'exécution).

Nous avons créé un mode de débogage dans notre interface afin d'aider l'utilisateur si un processus ne se déroule pas correctement. Cependant, ce mode n'est pas encore optimal et pourrait être complété. De plus, nous n'avons pas eu le temps de l'implémenter dans chacun de nos algorithmes.

Enfin, il est possible de lancer notre plugin par l'outil Macro d'ImageJ. Cependant, il subsiste une erreur : la procédure de création du *stack* d'images individuelles se déroule deux fois (une fois sur le *stack* de base et une autre sur le *stack* de particules sélectionnées). Ceci a pour effet d'afficher deux *stacks* finaux au lieu d'un seul. Résoudre ce problème permettrait d'avoir un déroulement optimal de notre programme depuis l'outil Macro.

Conclusion

Le but de notre projet était l'implémentation en Java ™ d'un outil pour le logiciel ImageJ permettant un piquage automatique de particules sur des images issues de cryo-microscopie.

Notre plugin se présente sous la forme d'une interface et donne à l'utilisateur le choix entre plusieurs algorithmes de piquage.

Nous avons implémenté trois algorithmes dans notre plugin : Différence de dilatation, Différence gaussienne et Corrélation d'images. Leur efficacité varie en fonction du type de particule à sélectionner et de la qualité de l'image à traiter.

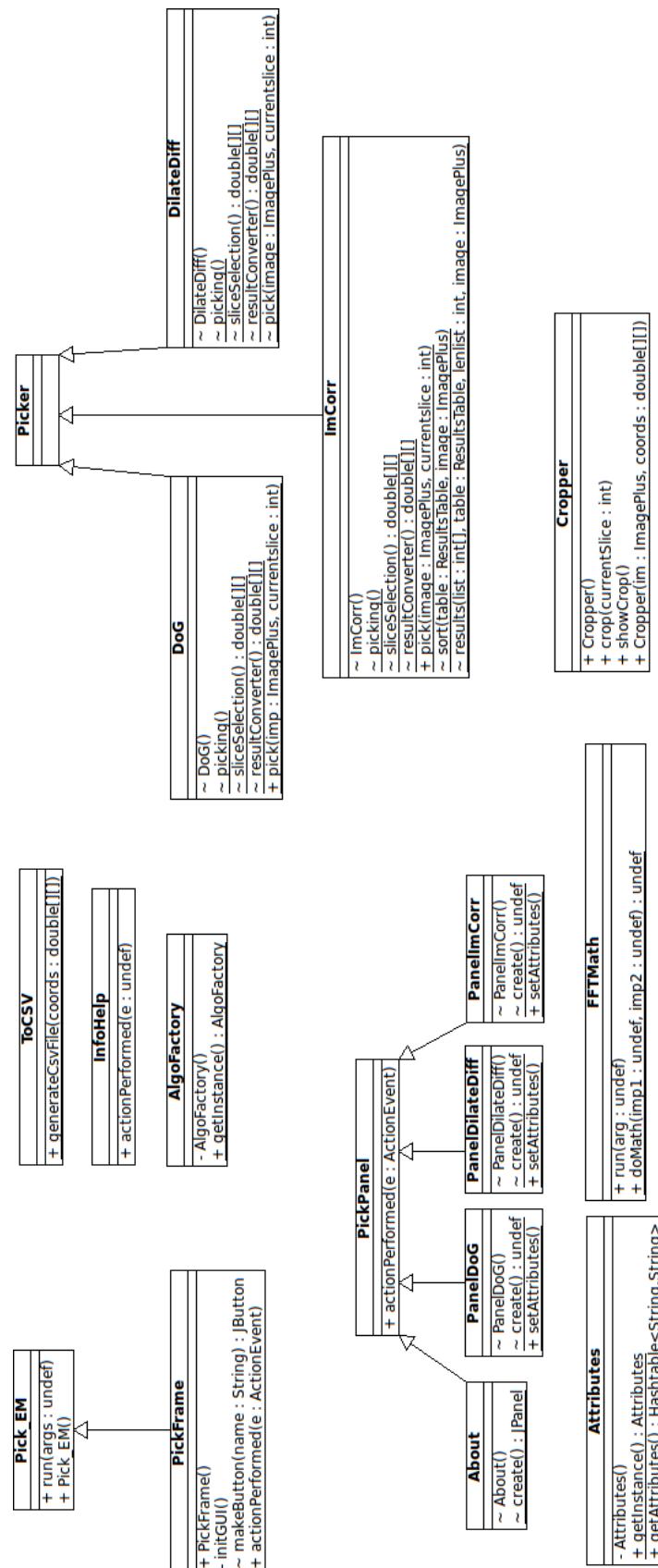
Cette expérience nous a permis de mieux maîtriser la programmation en langage Java ™ ainsi que de mieux connaître le fonctionnement du logiciel ImageJ. De plus, ce premier projet en équipe nous a donné un aperçu plus précis de ce que sera notre futur métier.

Bibliographie

- [1] Université de Bordeaux. Chimie et biologie des membranes et nanoobjets - umr 5248. <http://www.cbmn.u-bordeaux.fr>.
- [2] Inc. Free Software Foundation. Description de la general public licence. <http://www.gnu.org/licenses/licenses.fr.html>.
- [3] The Scripps Research Institute. Site web du scripps institute. <http://www.scripps.edu/>.
- [4] Robert Langlois, Jesper Pallesen, and Joachim Frank. Reference-free particle selection enhanced with semi-supervised machine learning for cryo-electron microscopy. *Journal of Structural Biology*, Jun. 2011.
- [5] National Institute of Health. National institute of health - site internet d'imagej. <http://www.rsbweb.nih.gov/ij/>.
- [6] Oracle. Java - langage de programmation orienté-objet. <http://www.oracle.com/technetwork/java/index.html>.
- [7] Oracle. Site internet d'oracle pour les archives jar. <http://docs.oracle.com/javase/tutorial/deployment/jar/>.
- [8] Phil Katz. PKWARE. Site internet de pkware pour les archives zip. <http://www.pkware.com/support/zip-app-note>.
- [9] William V. Nicholson and Robert M. Glaeser. Review : Automatic particle detection in electron microscopy. *Journal of Structural Biology*, Feb. 2001.
- [10] Wikipedia. Définition du langage orienté objet javascript. <http://en.wikipedia.org/wiki/Javascript>.
- [11] Wikipedia. Définition et description de la transformée de hough. http://en.wikipedia.org/wiki/Hough_transform.
- [12] Wikipedia. Définition et description de la transformée de hough généralisée. http://en.wikipedia.org/wiki/Generalised_Hough_transform.
- [13] Wikipedia. Définition et formules de la sensibilité et de la spécificité. http://fr.wikipedia.org/wiki/Sensibilite_et_specificite.
- [14] Yuanxin Zhu, Bridget Carragher, Fabrice Mouche, and Clinton S. Potter. Automatic particle detection through efficient hough transforms. *IEEE Transactions on Medical Imaging*, Sept. 2003.

Annexes

A.1 Diagramme des classes du plugin Pick_EM



A.2 Code source du plugin ImageJ Pick_EM