

Bertrand Provot  
3INFO

## Rapport de Stage

Extension d'un logiciel pédagogique  
de traitement d'image

**Maître de stage :** Julien Heulot

**Responsable pédagogique INSA :** Danièle Quichaud

**Lieu de stage :** Institut National de Sciences Appliquées de Rennes, Département EII

**Période de stage :** du 11 Juin 2018 au 27 Juillet 2018 puis du 20 Août 2018 au 31 Août 2018

# Table des matières

<b>I</b>	<b>Approche du stage, description des attentes</b>	<b>4</b>
1	Description de l'existant	4
2	Mission	5
3	Organisation et travail	5
4	Technologies utilisées	5
<b>II</b>	<b>Modifications apportées</b>	<b>6</b>
<b>5</b>	<b>Résolution des interruptions du logiciel</b>	<b>6</b>
5.1	Hough2 . . . . .	6
5.2	Séparation des plans (TSV ou RVB) . . . . .	6
5.3	Autres interruptions du logiciel . . . . .	7
<b>6</b>	<b>Corrections de bugs</b>	<b>7</b>
6.1	Grille de pixels . . . . .	7
6.2	Filtres . . . . .	8
6.3	Zoom . . . . .	10
6.4	Correctifs mineurs . . . . .	10
<b>7</b>	<b>Nouvelles fonctionnalités</b>	<b>10</b>
7.1	Uchar convertir . . . . .	10
7.2	Toolbar hider . . . . .	11
7.3	Classe de tests . . . . .	11
7.4	Refonte du menu . . . . .	12
<b>8</b>	<b>Traduction</b>	<b>12</b>
<b>III</b>	<b>Conclusion</b>	<b>13</b>
<b>9</b>	<b>Bilan et perspectives</b>	<b>13</b>
<b>10</b>	<b>Bilan personnel</b>	<b>13</b>

# Remerciements

Je tiens tout d'abord à remercier l'INSA qui me permet de réaliser ce stage au sein de ses locaux, et plus particulièrement au département EII et son personnel pour son accueil chaleureux ainsi que la mise à disposition du matériel et du bureau. Je tiens ensuite à remercier Luce Morin et Julien Heulot pour leur aide précieuse tout au long du stage et leur disponibilité. Enfin je souhaiterais remercier Marie Babel pour avoir fait suivre ma candidature à Luce pour ce stage. Merci enfin à Antoine Riotte pour les conseils, les recommandations et la bonne humeur.

# Introduction

Le stage se place dans le contexte du département EII qui possède son propre logiciel de traitement d'image, nommé ImageINSA également utilisé par d'autres entités notamment le département Informatique. Le logiciel ayant été constamment amélioré ces dernières années pour répondre aux attentes du département, il demandait certaines améliorations algorithmiques ou d'un point de vue interface.

Le stage avait donc lieu au sein du laboratoire de recherche du département EII à l'INSA Rennes, parmi le personnel du département. Nous étions deux stagiaires, Antoine Riotte (étudiant en troisième année à l'INSA Rennes au département EII au début du stage) et moi-même à travailler sur le même projet, dans le même bureau que nous avons partagé tout au long du stage. Nous restions malgré tout à proximité de nos interlocuteurs principaux qu'étaient Luce Morin et Julien Heulot lorsque nous avions besoin de conseils ou de faire le point sur nos avancées.

## Première partie

# Approche du stage, description des attentes

## 1 Description de l'existant

Dans un premier lieu donc, parlons du logiciel. Mon stage avait pour objectif la reprise du développement du logiciel open source ImageINSA utilisé notamment par l'INSA dans un cadre pédagogique, permettant de faire du traitement d'image. Le logiciel est utilisé aux départements Informatique et EII.

Sans rentrer dans les détails précis des fonctionnalités du logiciel, ce dernier possède plusieurs menus (voir figure ci-jointe), chacun permettant de réaliser des opérations sur des images que le logiciel ouvre. Les opérations vont de la visualisation d'histogrammes, à l'application de filtres en passant par diverses transformées ou méthodes d'analyse.

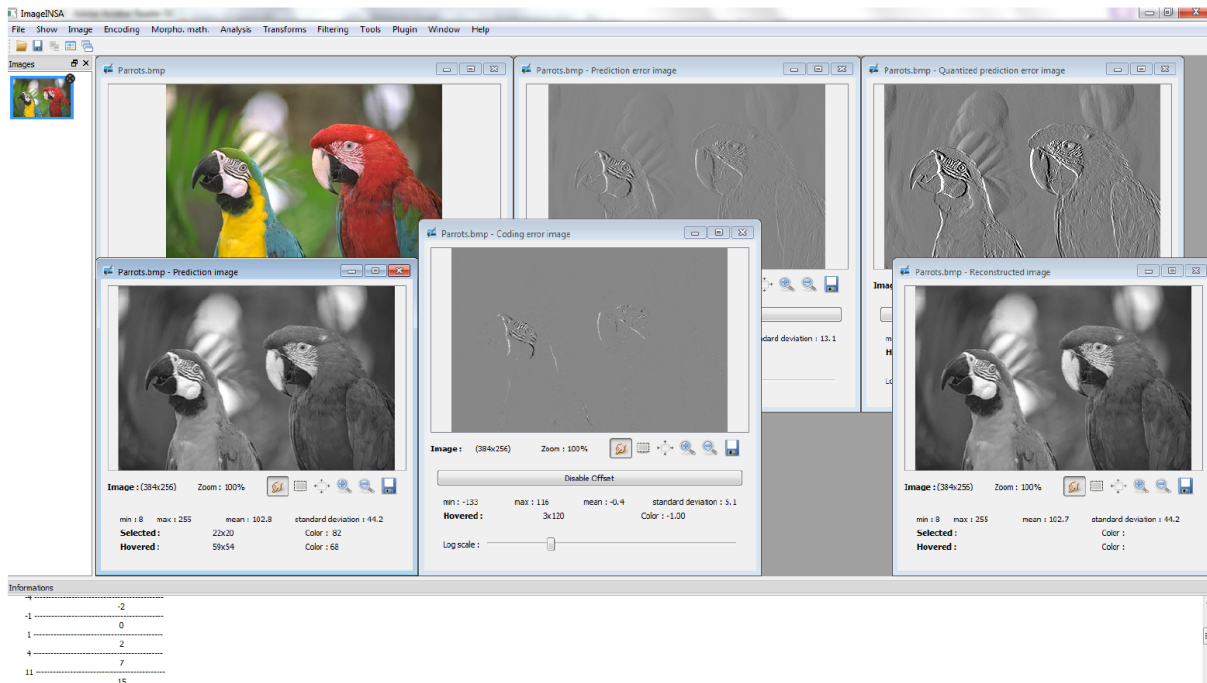


FIGURE 1 – Présentation du logiciel

ImageINSA est un logiciel qui voit le jour en 2012 en tant que projet d'élèves au sein du département info, basé sur un précédent logiciel nommé ImageEII, écrit en C sous visual studio. Lors de la création de ImageINSA, les algorithmes en C ont été conservés, mais l'interface a été entièrement reprise sous Qt et une nouvelle architecture logicielle a été développée en C++. Le logiciel ne cessera d'évoluer et de connaître des développeurs jusque cette année encore, dans un but d'amélioration constant et de répondre à de nouvelles exigences en permanence. Au début du stage, l'application est donc majoritairement fonctionnelle et aboutie mais possède malgré tout des bugs bloquants sur certaines fonctionnalités majeures.

Une des difficultés du stage résidait dans la complexité de l'architecture du logiciel; en effet qui dit développement sur plusieurs années dit changement de développeurs et améliorations nécessitant souvent une révision de l'architecture. Ni Antoine ni moi n'avons eu à changer l'architecture pendant le stage, le diagramme suivant est donc d'actualité au début comme à la fin du stage, les différences résidant dans l'ajout ou la suppression de certaines classes en des points spécifiques.

## 2 Mission

L'intitulé du stage étant le même pour Antoine et moi-même le premier jour, il a fallu préciser les tâches qui allaient nous incomber. Luce et Julien nous ont orienté vers nos tâches respectives le premier jour, la mienne résidant essentiellement dans la partie « visible » de l'application à savoir le développement Frontend du logiciel en Qt et sous l'environnement Linux dans une optique de rendre le logiciel compatible sur plusieurs plateformes. Comme Antoine travaillait sous l'environnement Windows nous avons donc régulièrement la possibilité de tester si les modifications apportées étaient compatibles sur les deux systèmes. Cette spécificité a d'ailleurs permis de découvrir des bugs qui étaient propre à Linux et Windows respectivement, et donc d'assurer la stabilité de l'application.

## 3 Organisation et travail

Comme évoqué, nous étions deux à travailler sur le même logiciel, parfois même à travailler sur l'édition d'une même classe, une organisation des plus rigoureuses était donc nécessaire. Dans un premier temps nous avons tout naturellement opté pour la reprise du versionning existant sous GitHub, lié à GitKraken pour une utilisation plus optimisée. Au début du stage, Julien nous a proposé d'expérimenter Glo, un utilitaire de gestion d'issues lié au repository Git d'ImageNSA, ce que nous avons fait et qui s'adapter particulièrement bien au stage qui finalement était de la résolution d'issues pour une bonne partie.

Ainsi après une journée d'installation du logiciel sur la machine avec toutes les librairies nécessaires et l'ajout de certaines lignes aux commandes nécessaire au lancement sous Linux, le logiciel était prêt et nous avons lancé une phase de test en utilisant la base des travaux de traitement d'Image (TP du département EII et Informatique), qui recoupée au tableur comportant toutes les erreurs recensées et ce que nous avons trouvé, nous a permis de dresser un issueboard<sup>1</sup> sous Glo, beaucoup plus lisible et interactif qu'un excel en local.

Une fois l'issue board dressé, le développement s'est fait en suivant une méthode qui ressemblait à de l'agile, je sélectionnais un issue ou un feature à développer, je me concentrais dessus et me donnais pour but de le finir dans le temps que j'estimais nécessaire à sa résolution. Le but était d'avoir résolu un bug ou ou fini de développer une feature importante en fin de semaine.

## 4 Technologies utilisées

Langage C++ : Le langage qu'utilise l'application, nécessaire à l'utilisation de Qt. Langage orienté objet, 95 % de mon travail était de l'édition de code écrit en C++ et headers ou templates ( .h et .tpp).

Qt : API orientée objet et développée en C++ qui offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution entre autres. Par certains aspects, elle ressemble à un framework lorsqu'on l'utilise pour concevoir des interfaces graphiques ou que l'on conçoit l'architecture de son application en utilisant les mécanismes des signaux et slots par exemple. (définition wikipédia)

Compilation : J'ai utilisé plusieurs utilitaires de compilation pour ce projet, notamment Cmake qui était utilisé pour faire les sources du projet, Qmake pour la classe de test que j'ai commencé à développer et make/makefiles pour créer les exécutables.

Gestion de version : Git qui est l'utilitaire de gestion de version le plus utilisé, utilisé sur la plateforme github qui héberge le dépôt du projet, ainsi que Gitkraken pour l'interface, l'arborescence et Glo pour l'issueboard.

Communication : Slack un service de messagerie lié au dépôt du projet qui permet de dialoguer mais aussi tenir informés tous les membres du projet sur Github de l'avancée du projet via l'envoi de messages automatiques quand des issues sont levés/résolus.

---

1. Tableau d'issues, un utilitaire permettant aux usagers de Git de relever les problèmes et de les signaler à tous

Résolution de bugs : Pour la résolution des bugs ils s'agissait assez fréquemment de fuites mémoires, assez compliquées à trouver dans une reprise de code, j'ai donc utilisé Valgrind qui est un outil très pratique permettant de trouver les fuites mémoires et autres problèmes d'allocation. L'outil est très puissant et permet d'avoir des diagnostics précis et détaillés, cependant parfois trop volumineux et difficiles à lire et interpréter. (voir annexe)

## Deuxième partie

# Modifications apportées

## 5 Résolution des interruptions du logiciel

Comme évoqué l'application possédait plusieurs problèmes et notamment des manipulations qui pouvaient provoquer des « plantages ». L'objectif premier résidait donc dans la résolution de ces erreurs qui empêchaient une utilisation complète du logiciel.

### 5.1 Hough2

Le logiciel possède une fonctionnalité appelée Hough2, qui permet l'application d'une transformée de Hough sur une image source. Sans rentrer dans les détails de l'algorithme car ils ne sont pas de mon ressort et non pertinents dans le cas de ce plantage, ce qu'il faut savoir est que l'utilisation de la fonction Hough2 génère une image double [référence], laquelle se trouve être dans un repère polaire. La fonctionnalité permet de choisir un pas pour l'affichage de Rho et de Théta (la base de coordonnées du repère résultat). Or pour un pas petit ( $<0.1$  pour Rho et  $<0.3$  pour Theta), l'application plantait complètement. Après un rapide diagnostique, j'en suis arrivé à la conclusion que les images générées étaient d'une taille beaucoup trop grande (en mettant 0.5 en pas de Théta, on doublait déjà la taille de l'image), et le logiciel ne pouvait pas les gérer.

La solution choisie pour palier au problème a été d'instaurer une valeur minimale de pas, même si ces conditions de plantage existent encore, elles sont beaucoup moins à même de se produire. La détermination du seuil minimal à choisir s'est faite à l'aide des observations expérimentales combinées d'Antoine et moi, après applications de la transformée sur des images référence.

### 5.2 Séparation des plans (TSV ou RVB)

Une autre fonctionnalité du logiciel est de permettre, pour une image couleur, de séparer ses plans de couleur et de générer trois images résultats respectivement associées aux couleurs rouge, verte et bleu, ou de générer les images doubles Teinte, Saturation et Valeur. De même il est possible avec l'option inverse de combiner des plans (TSV ou RVB) pour générer une nouvelle image.

Le logiciel avant correction permettait de séparer les plans RGB ou les plans TSV d'une image binaire ou en teintes de gris, ce qui ne servait à rien et amenait à des interruptions de l'application. Nous avons donc décidé d'éliminer cette possibilité, ce qui s'est fait simplement en supprimant l'option des menus pour les images qui pouvaient bloquer l'application.

De même pour la combinaison des plans, il existait la possibilité d'utiliser la fonctionnalité sans avoir aucun image d'ouverte mais cela entraînait des complications car on pouvait valider l'opération malgré l'absence d'image. Pour palier à cela, valider l'opération rend maintenant une erreur, et il est impossible d'aller plus loin pour l'utilisateur.

### 5.3 Autres interruptions du logiciel

Les autres plantages qui ont été relevés lors de l'examen préliminaires ne concernaient pas la partie frontend de l'application mais la partie algorithmique, cependant j'ai eu mon rôle à jouer dans leur résolution. Comme il s'agissait souvent de problèmes mémoires ou d'accès mémoires hors limites, il était difficile si ce n'est impossible de trouver les origines des problèmes sous Windows qui ne possède pas de console de debug (w7). J'aidais donc souvent à faire les diagnostics en exécutant Valgrind et en aidant à l'interprétation des résultats.

## 6 Corrections de bugs

Cette partie s'attache à la correction des bugs qui avaient pu être relevés, ou sur lesquels nous sommes tombés en utilisant l'application lors du stage. Il est important de noter que la résolution des bugs était ce qui occupait mon temps pour sa majeure partie.

### 6.1 Grille de pixels

Un des problèmes qui est rapidement sorti du lot était un problème d'affichage la grille de pixel. Cette dernière permet d'afficher un tableau qui à un pixel donné donne sa valeur sur 8bits (3 fois 8bits pour une image couleur, en ayant la possibilité de changer de champ R,G ou B). Le premier problème qui est ressorti pour cette fonctionnalité était le non-affichage de la dernière colonne et de la dernière ligne, qui parfois pouvait s'étendre à deux colonnes ou deux lignes. Le bug était particulièrement ardu à corriger, car identifié depuis longtemps, mais non résolu dans les précédents stages.

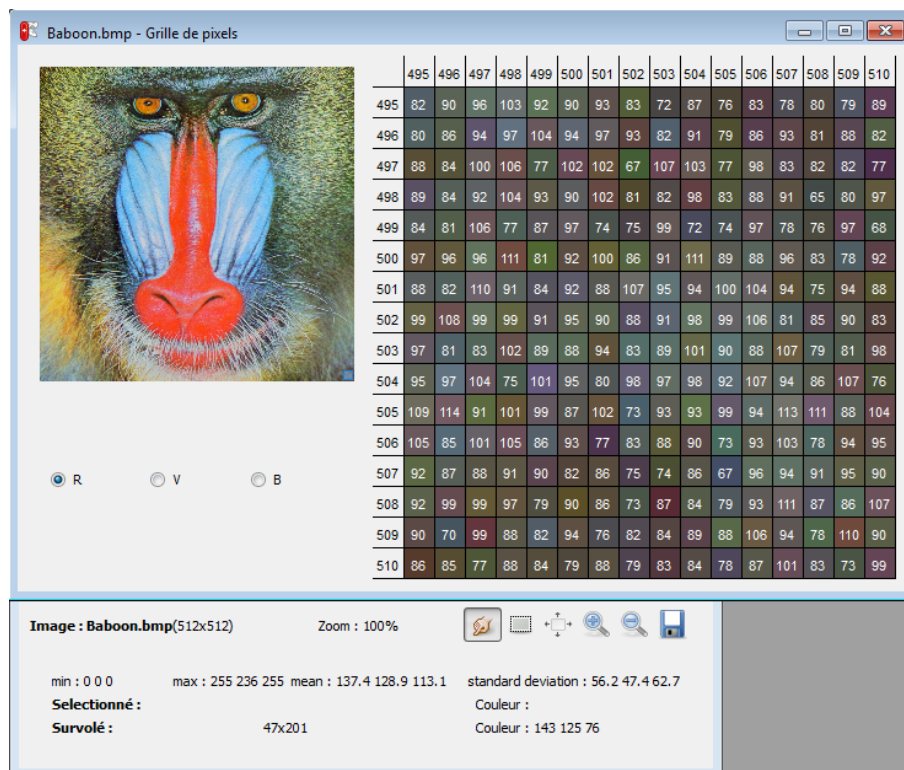


FIGURE 2 – Dernière ligne et colonne de la grille de pixel avant résolution du problème

Pour trouver le problème, il faut comprendre le fonctionnement de la grille de pixels et des choix d'affichage faits par les précédents développeurs. La grille de pixels est extraite à partir d'un carré de sélection visible une image. Une nouvelle image est ensuite créée, qui fait permet la visualisation des pixels



zoomés. Chaque pixel de l'image source possède une nouvelle taille, également en pixel, sur la grille. Cette taille est de 25\*25 pour une image uChar et 32\*32 pour une double. Le problème est que la grille dessinée ne prend pas en compte la marge de la fenêtre. On se retrouve donc avec une voire deux lignes ou colonnes qui sont virtuellement affichées au moment du dessin de la grille de pixels mais qui sont cachées derrière la marge de la fenêtre.

```
void PixelGrid::resizeEvent(QResizeEvent* event) {
    QSize* evtSize = new QSize();
    evtSize->setWidth((event->size().width()-PIXEL_S)/PIXEL_S);
    evtSize->setHeight((event->size().height()-PIXEL_S)/PIXEL_S);
    emit resized(*evtSize);
}
```

Le problème se corrige donc via ce code qui spécifie un signal à envoyer lorsque la fenêtre change de taille, dont la valeur ne prenait pas en compte ce qui était affichable auparavant.

Un autre problème, moins gênant mais malgré tout peu esthétique était le fait que des carrés blancs apparaissaient lorsque l'on étirait la fenêtre alors que celle-ci était maximale. Une fois ce problème réglé, un autre en découlait à savoir le fait que refermer la fenêtre conduisait à un affichage hors-limite, ce que j'ai également réglé. Ces problèmes étaient dû au fait que à chaque fois l'agrandissement de la fenêtre ne savait pas si la dernière ligne ou colonne était déjà affichée (deux premiers schéma de la figure). Désormais quand la dernière ligne ou colonne est déjà affichée, la zone de sélection de la grille de pixel dessinée s'agrandit vers l'intérieur et non plus vers l'extérieur. (dernier schéma de la figure)

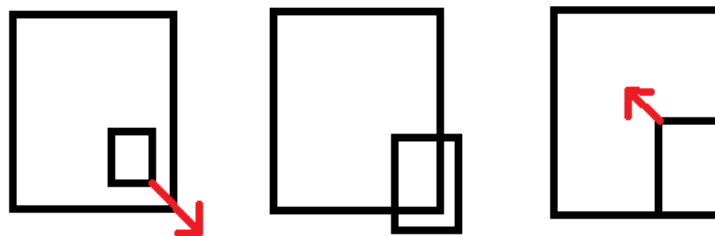


FIGURE 3 – Représentation du problème graphique

## 6.2 Filtres

Une des fonctionnalités majeures du logiciel est la possibilité d'appliquer des filtres sur une image. Les filtres permettent l'application de transformations sur les pixels d'une image en fonction de leur environnement proche. Ils nécessitent plusieurs informations de la part de l'utilisateur : le fichier dans lequel se trouve le filtre, le type du filtre et le type de l'image résultante entre autres.

Dans une premier temps donc, je me suis attaché à régler l'édition de filtres. Le menu n'était pas complètement stable, notamment un ascenseur qui affichait des valeurs erronées, et le menu nécessitait l'implémentation de nouvelles fonctionnalités notamment pouvoir choisir soi-même le fichier que l'on souhaite utiliser, dans l'arborescence fichier. En effet, jusque-là l'utilisateur ne possédait qu'une seule possibilité de fichier de filtres, placé dans un emplacement par défaut. J'ai donc ajouté la possibilité de choisir à la fois où et quel fichier choisir lorsque l'on crée un filtre. Enfin j'ai rajouté des blocages pour empêcher les bugs, afin que l'on ne puisse pas valider l'opération sans avoir au préalable choisi un fichier au format valide par l'utilisation d'expressions régulières. J'ai également pu ajouter la possibilité d'enlever la normalisation des filtres (qui était automatique mais caché auparavant).

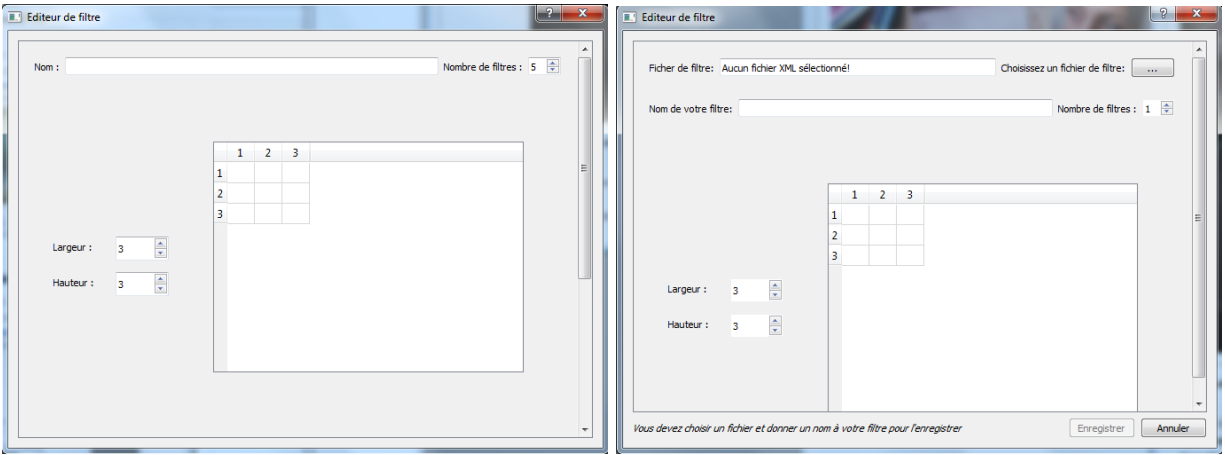


FIGURE 4 – Comparaison de l'ancien menu d'édition de filtres (gauche) avec le nouveau (droite)

Dans un second temps je me suis attaché au menu d'application des filtres, en séparant notamment l'interface en deux pour la sélection des filtres personnalisés.

Les fonctionnalités étaient développées mais il restait des points à éclaircir dans la fenêtre d'application des filtres. Le menu déroulant affiche désormais correctement le nombre de filtres sélectionné par l'utilisateur, l'affichage est désormais stable, l'utilisateur peut désormais chercher le path vers son fichier de filtre sans avoir à utiliser la filtre standard (qui ne se trouvait pas forcément dans une zone où l'utilisateur a les droits en écriture, notamment en TP). Le choix du fichier de filtre pouvant interrompre l'application, j'ai pris grand soin de lever des exceptions si l'utilisateur choisit des fichiers non conformes ou ne choisit rien. Enfin j'ai pu rajouter des fonctionnalités (notamment l'ajout d'un décalage et d'une mise à l'échelle de l'image générée en accord avec Antoine qui s'occupait de l'algorithme.

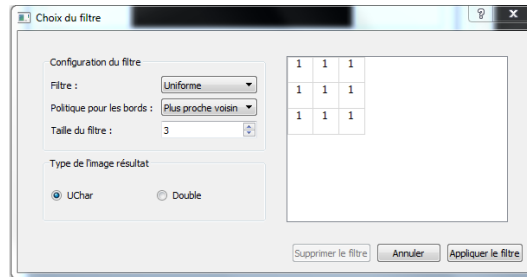


FIGURE 5 – Ancien menu d'application des filtres

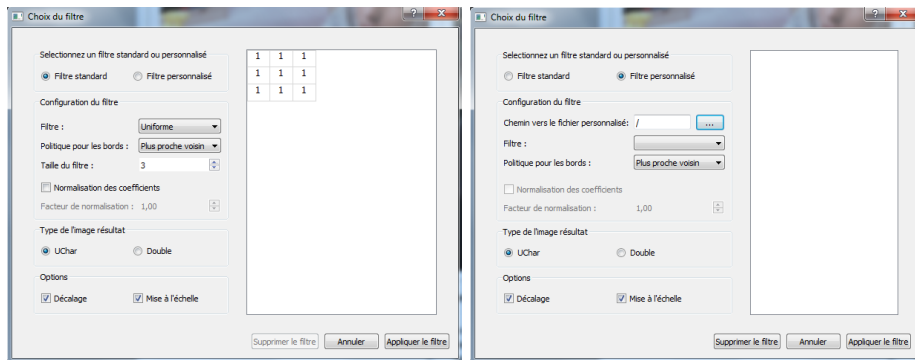


FIGURE 6 – Nouveau menu d'application des filtres (filtre standard et personnalisé)

## 6.3 Zoom

Le logiciel travaillant sur des images, il possède bien évidemment une option permettant le zoom avant/arrière sur les images sur lesquelles ont travaillé. Cependant à la réception du logiciel, un problème de coordonnées existait lorsque l'on travaillait sur une image agrandie ou rétrécie.

Le problème vient simplement du fait que le précédent développeur de cette fonctionnalité a voulu mettre en place un système qui n'affiche plus les coordonnées lorsque le curseur se trouve en dehors de l'image, mais le test qu'il faisait n'était pas le bon. Ainsi j'ai temporairement remplacé ce test par un `true`, ce qui fait que le logiciel affiche désormais les coordonnées en dehors de l'image (mais évidemment sans rendre de valeur pour ces pixels imaginaires). Je n'ai cependant pas réussi à réaliser ce que le précédent développeur souhaitait faire.

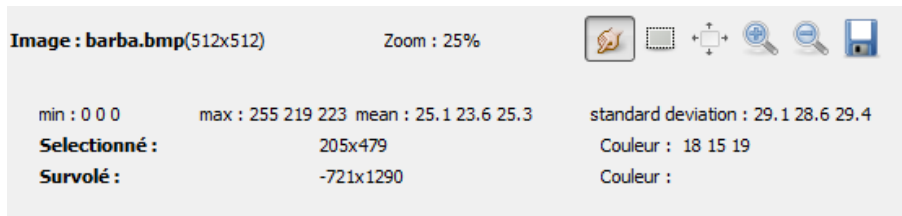


FIGURE 7 – Visualisation des pixels hors images dans le champ survolé

## 6.4 Correctifs mineurs

Une partie importante de mon travail a en réalité porté sur une myriade de petites corrections, pouvant simplement permettre l'affichage de logs plus propres et traduits, clarifier le nom des fenêtres lorsque celles-ci sont générées par les différentes opérations du logiciel, ou bien corriger des bugs d'interface plus triviaux.

A titre d'exemple, griser des boutons d'une interface se fait très simplement grâce à Qt, le tout est de trouver le bouton en question et d'utiliser la fonction pré-implémentée `setEnabled(bool)`, que possèdent tous les widgets dans Qt. Qt étant une API orientée objet il suffit d'appeler `myButton.setEnabled(bool)` et le tour est joué. Cela me permet d'aborder le fait que Qt est extrêmement complet, si bien que l'on peut virtuellement tout faire à condition d'avoir la documentation sous la main. Les premières semaines de mon stage se sont d'ailleurs faites avec la documentation Qt ouverte en permanence, car je ne connaissais pas l'API et tout ce qu'elle pouvait offrir..

Enfin j'ai eu l'occasion de refondre en partie l'interface du logiciel, notamment les places et catégories des menus ainsi que les opérations qu'ils contiennent. Cela s'est fait directement dans le main en créant ou ajoutant les opérations directement aux menus d'opérations (couche récente) ou en modifiant les services qui viennent greffer des menus Qt aux menus d'opérations déroulants.

# 7 Nouvelles fonctionnalités

## 7.1 Uchar converter

Une de mes tâches a été le développement d'une fonctionnalité permettant de transformer une image double en image simple. L'algorithme n'était pas de mon ressort, mais le dialogue et la fenêtre sont mes œuvres. Le développement de cette nouvelle fonctionnalité s'est fait par la création d'une classe héritant d'opération (`UcharConvertOp`). Ce qu'il est intéressant de noter est que je n'ai pas utilisé Qt Designer qui est un outil de la suite Qt permettant la création intuitive de dialogue ou fenêtre, j'ai préféré coder dans une classe de dialogue les fonctionnalités de la fenêtre car j'étais plus à l'aise avec cette méthode, et qu'elle me semble plus flexible aux améliorations ultérieures.

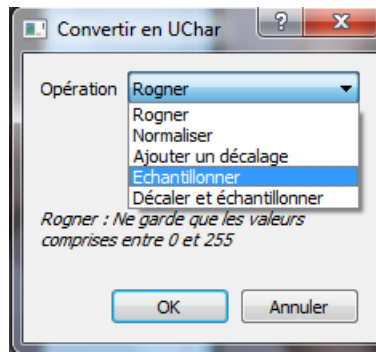


FIGURE 8 – Fenêtre de conversion en Uchar

## 7.2 Toolbar hider

Après utilisation répétée de l'application, on s'est rendu compte que si l'on cachait la barre de visualisation d'images latérale, on ne pouvait pas la rouvrir ce qui était assez problématique. J'ai donc décidé de rajouter un petit bouton très simple dans la barre de menu qui permet de cacher et afficher la barre d'images latérale (figure du bouton).

## 7.3 Classe de tests

Une des tâches majeurs du stage a été le développement d'une classe de test dont mon approche a été de séparer les tests algorithmiques des tests d'interface. La classe de test que j'ai réalisée concerne une partie de l'interface du logiciel. Tous les tests suivent la même idée : on clique virtuellement sur les boutons de manière à générer les images résultats. L'idée est ensuite de comparer via une seconde classe de test les images résultats à des images références (soit faites sous un logiciel tiers comme c'est le cas actuellement pour le dossier ref qui contient des images issues d'opération faites sous matlab, soit faites avec ImageNSA mais validées).

Le dossier test contient un exécutable test qui lance la génération d'images pour les opérations implémentées, dans trois dossiers d'images : ref, src et res. Ref contient les images références à partir d'image sources placées dans src. Res est le répertoire où les images sont générées par la classe de test.

Pour générer l'exécutable via qmake, une certaine marche à suivre s'impose. Pour cela j'ai rédigé un readme pas à pas (voir Annexe) car cela a pris un certain temps pour générer un exécutable fonctionnel. Enfin, une méthode de comparaison d'image est présente dans la classe de test, mais n'a pas été testée suffisamment.

L'idée du fonctionnement de la classe de test suit le code ci-dessous :

```
void TestGui::testQuantifOpNonLinearMean3(){
    WindowService* ws = dynamic_cast<WindowService*>
        (gi->getService( GenericInterface::WINDOW_SERVICE));
    ws->addFile( QString("src/test_quantization_berber.bmp") );

    std::vector<GenericOperation*> opvect = _image->getOperations();
    std::vector<GenericOperation*>::iterator it;

    for (it = opvect.begin(); it != opvect.end(); ++it){
        if ((*it)->getName()=="Quantification"){
            (dynamic_cast<QuantificationOp*>(*it))->setTest(true);
            (dynamic_cast<QuantificationOp*>(*it))->setQuantif(2);
            (dynamic_cast<QuantificationOp*>(*it))->setValues(3);
        }
    }
}
```

```

    (*it)->operator()(ws);
  }
}

```

On commence par ouvrir virtuellement la fenêtre en simulant des ouvertures d'images. On appelle ensuite l'opération à tester en insérant directement dans la fenêtre les valeurs que l'on souhaite. L'image est donc générée.

Cela ne marche cependant que pour les opérations (qui font partie de la couche récente de l'application), pour les couches plus basses une classe de test existe déjà, mais je n'ai pas réussi à la faire fonctionner par manque de documentation. La classe de test comporte une dizaine d'opérations à la fin de la période de stage.

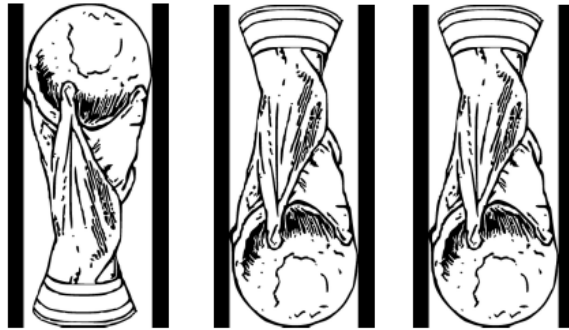


FIGURE 9 – Trois images utilisées pour les tests, respectivement dans les dossiers Src, Res et Ref

## 7.4 Refonte du menu

La tâche par laquelle j'ai fini mon stage a été la refonte du menu de l'application. La tâche n'était pas difficile en soi vu qu'il ne s'agissait que de modifier l'ordre dans lequel les opérations et différents widgets sont ajoutés au menu. La partie intéressante reposait sur la réflexion et la prise en compte de l'ergonomie de l'application pour bien penser où sont sensés être les différents menus, avec notamment une phase de réflexion avec Luce Morin et Veronique Coat, en prenant en compte les avis et opinions de chacun sur la position des diverses opérations.

## 8 Traduction

Une partie du stage a porté sur le fait de rendre aussi international que possible le logiciel. Une partie du logiciel était déjà traduite mais il persistait malgré tout des erreurs dans les traductions existantes ou simplement des oublis. Pour les traductions, le développement en Qt est parfait car il possède des fonctionnalités pré implémentées.

Si un projet nécessite une traduction, Qt peut générer des fichiers au format .tr qui doivent être compilés en binaire avant de pouvoir être liés et utilisés par le projet lors du make. Ces fichiers de traduction permettent pour les langages choisis par le développeur (ici Anglais Américain et Français) d'associer des traductions à des Qstring « marqués ». Ce que je nomme Qstring marqués sont les strings utilisés dans les sources c++ générés par tr(string) ou bien QApplication->translate(string), les deux étant sensiblement similaires. « Marquer » les strings est la seule partie du processus de traduction nécessitant de toucher au code, le reste se faisant avec les fonctionnalités de Qt.

D'abord il me faut ouvrir un terminal, me rendre dans mon répertoire où sont générés les fichiers de traduction (il y en a deux dans ce logiciel du fait des développements par couches superposés) et lancer la

commande `lupdate ../* -ts *.ts` Ceci a pour effet de signaler aux `.tr` que de nouveaux strings sont marqués et nécessitent une traduction.

Ensuite je lance la commande `linguist *.ts` qui a pour effet d'ouvrir dans le logiciel QtLinguist tous les fichiers `.tr` (un par langue), et de donner une interface lisible dans laquelle il ne reste plus qu'à traduire les strings en question, dans les langages nécessaires.

Une fois toutes les traductions terminées et validées, il ne reste plus qu'à lancer la commande `lrelease *.ts` qui génère les binaires associés aux traductions en `.qm` qui mettront à jour de manière efficaces toutes les traductions lors du prochain `make`.

## Troisième partie

# Conclusion

## 9 Bilan et perspectives

Au terme du stage, l'application est quasiment intégralement stable, ce qui était le but premier du stage à savoir qu'il n'y ait plus d'interruptions bloquantes sur le logiciel. En second lieu la quasi intégralité des bugs ont été corrigés sur le tableur qui nous servait de référence pour les modifications à apporter. L'application bénéficie de plus de fonctionnalités qu'elle n'en avait en Juin, aussi bien d'un point de vue algorithmique qu'interface, comme le rapport l'explique. La centaine de commits que j'ai pu faire de mon côté aura une utilité qui sera validée par l'année 2018/2019 et ce que les utilisateurs relèveront de leur côté lors des TP dans les département EII et Informatique, les mêmes que nous avons utilisés pour voir les tares de l'application au début du stage.

D'un point de vue plus technique il reste cependant de nombreux points d'amélioration, notamment l'intégralité de la classe de test à finir, sur la base de ce que j'ai commencé. En effet ce que j'ai commencé à faire fonctionne mais la masse de travail était assez importante, aussi j'apporte une approche solide mais à finir. La comparaison des images générées est à créer de toute pièce par exemple.

L'objectif de stabilité cross plateforme est quant à lui un succès, avec un logiciel qui semble stable sur les deux plateformes vis à vis de l'utilisation, des différents essais et des tests que nous avons pu faire dessus. Les TP et le temps nous diront si c'est bel et bien le cas.

Enfin, après deux mois d'utilisation et de développement de l'application je peux valider le besoin de refondre l'architecture de l'application, qui n'est plus du tout ergonomique, notamment l'utilisation du submodule git pour la librairie Detiq-t qui au final n'est utilisée que par l'application ImageINSA, une fusion est donc très largement envisageable.

## 10 Bilan personnel

Ce stage a été pour moi une expérience d'un intérêt immense, aussi bien d'un point de vue technique qu'humain.

J'ai pu apprendre à développer en C++, langage que je ne connaissais que très peu et j'ai pu commencer à maîtriser l'API Qt, et notamment me rendre compte de sa puissance, sa versatilité et sa polyvalence. La méthode de travail en issue board qui se rapprochait de l'agile m'a beaucoup plu et je pense opter pour une méthode similaire dans mes projets futurs.

Le travail au sein d'un laboratoire de recherche m'a aussi permis de me faire une idée de ce sur quoi le diplôme ouvre d'un point de vue professionnel, et m'a ouvert à la réflexion sur une thèse.

# Annexes

## Annexe 1 : rapport type de valgrind après application sur le logiciel

```
==18606== Invalid write of size 4
==18606==    at 0x4FACD5: void imagein::Histogram::computeHistogram<unsigned
    char>(imagein::Image_t<unsigned char> const&, unsigned int, imagein::
    Rectangle const&) (in /home/bprovot/Documents/eiimage/build/ImageINSA/
    imageinsa)
==18606==    by 0x4FABB4: imagein::Histogram::Histogram<unsigned char>(
    imagein::Image_t<unsigned char> const&, unsigned int, imagein::Rectangle
    const&, int) (in /home/bprovot/Documents/eiimage/build/ImageINSA/
    imageinsa)
==18606==    by 0x4F346A0: genericinterface::GenericHistogramView::
    GenericHistogramView(imagein::Image_t<unsigned char> const*, imagein::
    Rectangle, bool, int, bool, bool, bool, bool) (in /home/bprovot/Documents
    /eiimage/build/ImageINSA/lib/libcore.so)
==18606==    by 0x4F3FAFB: genericinterface::HistogramView::HistogramView(
    imagein::Image_t<unsigned char> const*, imagein::Rectangle, bool) (in /
    home/bprovot/Documents/eiimage/build/ImageINSA/lib/libcore.so)
==18606==    by 0x4F3FECE: genericinterface::HistogramWindow::
    HistogramWindow(imagein::Image_t<unsigned char> const*, imagein::
    Rectangle, QString, bool) (in /home/bprovot/Documents/eiimage/build/
    ImageINSA/lib/libcore.so)
==18606==    by 0x4F49955: genericinterface::ImageWindow::showHistogram() (
    in /home/bprovot/Documents/eiimage/build/ImageINSA/lib/libcore.so)
==18606==    by 0x4F1E0BE: genericinterface::UtilityService::showHistogram()
    (in /home/bprovot/Documents/eiimage/build/ImageINSA/lib/libcore.so)
==18606==    by 0x4F5AC6D: genericinterface::UtilityService::
    qt_static_metacall(QObject*, QMetaObject::Call, int, void**) (in /home/
    bprovot/Documents/eiimage/build/ImageINSA/lib/libcore.so)
==18606==    by 0x6930D29: QMetaObject::activate(QObject*, int, int, void**)
    (in /usr/lib/x86_64-linux-gnu/libQt5Core.so.5.5.1)
==18606==    by 0x53E7411: QAction::triggered(bool) (in /usr/lib/x86_64-
    linux-gnu/libQt5Widgets.so.5.5.1)
==18606==    by 0x53E9897: QAction::activate(QAction::ActionEvent) (in /usr/
    lib/x86_64-linux-gnu/libQt5Widgets.so.5.5.1)
==18606==    by 0x53E9E1E: ??? (in /usr/lib/x86_64-linux-gnu/libQt5Widgets.
    so.5.5.1)
```

## Annexe 2 : Readme pour l'aide à la compilation de la classe de test

### ## - COMPILATION

- In order to launch the test class you need to follow these steps :
  - First of all compile the project with CMAKE (or whatever compiler you're using) in your build directory, then build the .exe with make.
  - Then go back to this directory and use qmake —project.
  - qmake
  - make
- You will probably have to edit the .pro file before the qmake in order to specify the path to the qwt library in the include path (e.g /mydirectory /qwt6.0.1/src), and to specify the .a used by the test class to run the tests by adding the line "LIBS += -L../build/ImageINSA/lib llibimageinsa" to the .pro file.
- The test directory is composed of three sub directories named ref, res and src. Res should be empty at the end of your compilation, because it contains the resulting image after application of the operations. Src contains all the source which the test are applied to. Ref contains the images reference with each operation applied on through another software (here matlab).
- Everytime you execute the test you have to run a make distclean, then do the qmake, make and LD\_LIBRARY\_PATH=/home/bprovot/Documents/eiimage/build /ImageINSA/lib ./test, in order to empty the src directory and not run test on a previous value of the resulting images.

### ## - GENERATING REFERENCE IMAGES

- We used MATLAB for references. To do so we had to use matlab function trying to mimic imageINSA functions, that can be found in the MATLAB folder.
- To open a .vff image in MATLAB there is a function called vff.m findable in the so called MATLAB folder. It needs the path to the image, if you give no parameter it will open a window to select the image.



## Résumé

Dans le cadre de mon cursus en école d'Ingénieur Informatique à l'INSA de Rennes, j'ai été amené à travailler le temps d'un stage au sein du département EII de la même INSA sur le développement d'un logiciel à but pédagogique.

Le logiciel appelé ImageINSA est un logiciel de traitement d'image qui a progressivement évolué au fur et à mesure des différents stages et projets qui y touchaient ces dernières années. Dans cette optique d'évolution constante je suis intervenu sur son développement dans le but de corriger les bugs, interruptions et autres problèmes qui pouvaient intervenir lorsque le logiciel fonctionnait et qui empêchaient son utilisation complète. Ma mission consistait également à améliorer l'interface de l'application.

Dans une seconde période j'ai entamé l'écriture d'une classe de tests, pour rendre l'application plus facile à modifier, en gardant toujours en tête qu'elle sera sûrement encore modifiée par des développeurs.

## Abstract

As part of my studies in Computer Science Engineering at INSA Rennes, I did an internship in the Industrial Electronics and Informatics department of the INSA Rennes. The internship had for subject the development of a teaching oriented software.

The software is called ImageINSA and is used to do Image Processing. It has progressively evolved during the past years as other interns or projects worked on it. In this optic of constant evolution, my mission was to fix the remaining bugs, crashes or other problems that could happen when the software was running and that prevented a complete use of it. I also had to improve the user interface of the application.

On a second time I started to write a test class in order to make the software easier to modify, keeping in mind the application will probably keep being upgraded in the years to come.