

Extension d'un logiciel pédagogique de traitement d'images

Lieu et dates du Stage

INSA de Rennes : département Électronique et Informatique Industrielle (EII)

Du 12 juin 2017 au 28 juillet 2017 puis du 28 août 2017 au 1^{er} septembre 2017

Tuteur du stage

Luce Morin

Correspondant pédagogique INSA Rennes

Bruno Arnaldi

Sommaire

Introduction	4
1 Présentation du logiciel ImageINSA	5
1.1 Historique	5
1.2 Fonctionnement/Architecture	5
1.3 Environnement de développement	6
2 Réalisation technique/pratique	7
2.1 Compilation du logiciel	7
2.1.1 Installation de l'environnement logiciel	7
2.1.2 Réalisation d'un script pour la compilation du logiciel	7
2.2 Gestion du projet sous git	8
2.3 Extension du logiciel	9
2.3.1 Ajout des boutons « sauvegarder » et « zoom »	9
2.3.2 Ajout bouton suppression normalisation	10
2.3.3 Résultats Double opérations images standards	10
2.3.4 Correction de l'affichage de l'élément structurant	11
2.3.5 Traduction français/anglais	11
2.4 Correction des algorithmes	12
2.4.1 Inventaire des corrections à apporter	12
2.4.2 Correction entropie	12
2.4.3 Correction transformée de Hough	12
2.4.4 Ajout profil ligne/profil colonne	13
2.4.5 Correction projection horizontale/verticale	14
Conclusion	15
A Script pour la compilation du logiciel ImageINSA	18
B Mode d'emploi pour la compilation du logiciel sur la machine virtuelle (x86)	19
C Planning	21
D Tableau récapitulatif des bugs restants	22
E Tableau récapitulatif des améliorations restantes	24

Remerciements

Je tiens à remercier Luce Morin pour m'avoir offert l'opportunité de réaliser ce stage ainsi que pour ses précieuses informations sur les algorithmes de traitement d'images.
J'aimerais également remercier Alexandre Sanchez pour ses conseils techniques concernant la compilation du logiciel ImageINSA ainsi que la gestion du projet sur le logiciel git.
Enfin j'aimerais remercier toute l'équipe du département EII pour l'accueil et la sympathie dont elle a fait preuve à mon égard.

Introduction

J'ai réalisé mon stage d'été au département Électronique et Informatique Industrielle (EII) de l'INSA de Rennes.

L'objet du stage était d'améliorer le logiciel ImageINSA, qui est un logiciel de traitement d'images développé à l'INSA Rennes et utilisé par les étudiants du département EII¹ dans le cadre de travaux pratiques. Même s'il existait déjà une version fonctionnelle du logiciel, cette dernière comprenait plusieurs bugs et les enseignants souhaitaient ajouter des fonctionnalités pour améliorer l'ergonomie. Les objectifs de ce stage étaient donc d'apporter des modifications à l'interface graphique afin de la rendre plus ergonomique mais aussi de corriger certaines erreurs apparaissant dans les algorithmes de traitement d'image et d'ajouter de nouvelles fonctionnalités. Enfin, un autre objectif était de simplifier la maintenance du logiciel et notamment sa compilation. J'ai donc également eu l'occasion de prendre en main le processus de compilation du logiciel ainsi que l'installation de l'environnement nécessaire à son fonctionnement, ce qui m'a permis de mieux comprendre son architecture.

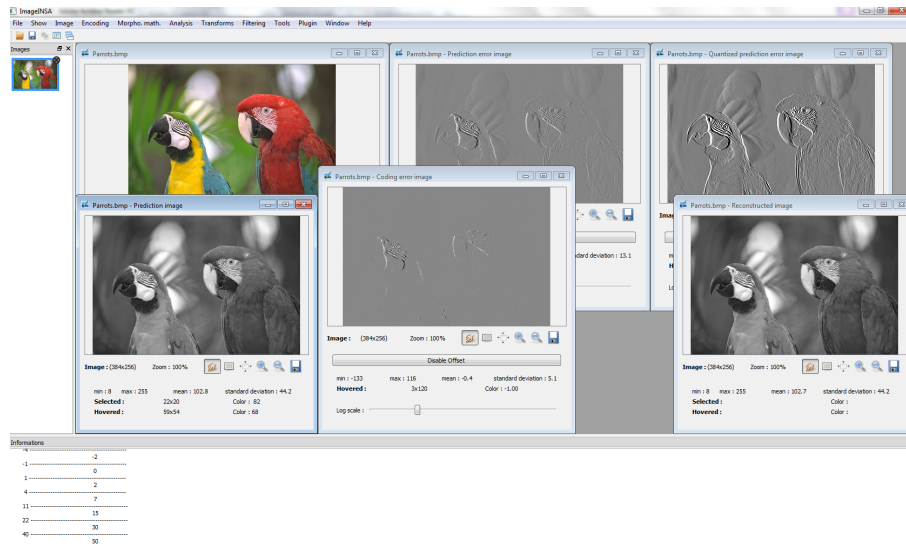


FIGURE 1 – Capture d'écran du logiciel ImageINSA

1. Électronique et Informatique Industrielle

Chapitre 1

Présentation du logiciel ImageINSA

1.1 Historique

ImageINSA est à l'origine né sous un autre nom : ImageEII. Cette première version du logiciel, était basée sous l'environnement de développement de Microsoft Visual C++ 6. L'inconvénient de celle-ci était qu'elle n'était pas compatible avec d'autres systèmes d'exploitation tels que linux ou macOS par exemple. De plus, ces technologies ayant été rendues obsolètes par Microsoft, il a été décidé de mettre en œuvre une refonte du logiciel en utilisant des technologies plus modernes. C'est ainsi qu'une nouvelle version est née et le logiciel fut rebaptisé EIImage puis ImageINSA pour l'occasion. Cette nouvelle version est développée avec le langage C++ et Qt, un framework utile pour concevoir, entre autres, des interfaces graphiques de manière intuitive.

Au fil du temps, ImageINSA a été amélioré[2] jusqu'à obtenir une version qui est celle actuellement utilisée pour les travaux pratiques des étudiants du département EII. Cependant, cette version a quelques soucis d'ergonomie et plusieurs erreurs sont présentes dans les différents algorithmes, ce qui pose problème d'un point de vue pédagogique. De plus, certaines fonctionnalités manquent encore à l'appel.

1.2 Fonctionnement/Architecture

Le logiciel ImageINSA est basé sur une librairie appelée Detiq-t[3]. Cette dernière, développée par des étudiants du département informatique de l'INSA de Rennes dans le cadre d'un projet logiciel de 4^e année, se compose de deux grandes parties :

L'interface générique : qui sert de base pour l'interface graphique du logiciel ImageINSA.

La librairie ImageIn : qui contient les algorithmes de base du traitement d'image.

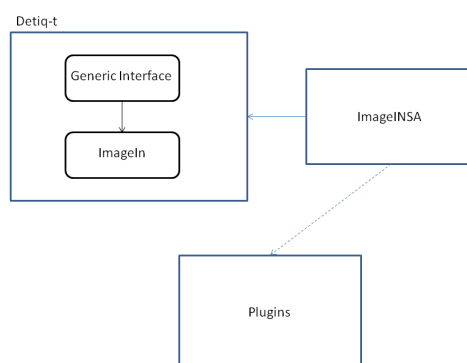


FIGURE 1.1 – Architecture du logiciel ImageINSA

Cette librairie est très utile car elle permet par exemple de s'appuyer sur ce qui est présent dans l'interface générique pour ajouter plus facilement d'autres éléments à l'interface graphique du logiciel. De même on peut se servir des fonctions élémentaires de la librairie ImageIn pour réaliser des algorithmes plus complexes par la suite.

1.3 Environnement de développement

Le développement du logiciel s'effectue sur Windows 7. ImageNSA utilise les bibliothèques suivantes :

Qt : qui est une Interface de Programmation Active (API) orientée objet en C++ permettant entre autres de créer et de gérer une interface graphique, mais qui offre aussi des composants d'accès aux données, de connexions réseaux, etc.

qwt : qui est une bibliothèque supplémentaire pour Qt permettant entre autres d'afficher des graphiques en 2D.

RandomLib : une bibliothèque permettant de générer des nombres aléatoires avec précision.

Zlib : permettant la compression/décompression de fichiers.

Jpeg et LibPng : contenant des outils utiles pour les images au format Jpeg et Png.

Chapitre 2

Réalisation technique/pratique

Nous verrons tout d'abord le processus de compilation du logiciel avant de s'intéresser à la gestion du projet sous GitHub puis enfin nous passerons aux fonctionnalités ajoutées et aux corrections apportées aux algorithmes.

2.1 Compilation du logiciel

2.1.1 Installation de l'environnement logiciel

Mon premier objectif au commencement du stage était l'installation de l'environnement du logiciel et sa compilation. Je me suis dans ce but aidé du guide de compilation présent sur le dépôt GitHub du logiciel[1]. J'ai dans un premier temps réalisé ceci sur une machine virtuelle déjà existante reprenant les caractéristiques techniques de machines présentes en salle de travaux pratiques, puis je suis ensuite passé sur ma machine de travail réelle. Comme précisé plus haut, ImageINSA a besoin de la bibliothèque Detiq-t ainsi que des librairies indiquées dans la section 1.3 pour fonctionner. Tout cela semblait de prime abord être assez rapide à installer, cependant une des contraintes majeures qui m'était donnée était de travailler sous Windows, et certains programmes tels que qwt et RandomLib devant être installés en ligne de commande, cela a quelque peu compliqué les choses. Car en effet je n'avais pas eu réellement l'occasion d'utiliser le terminal de Windows auparavant, et celui-ci fonctionne différemment de celui de linux. En effet, certaines commandes sont impossibles à utiliser à moins d'être ajoutées au PATH, ce à quoi il faut s'adapter. Par exemple, pour compiler et installer qwt, il faut utiliser les commandes *qmake* et *qmake install* de Qt. Le problème étant que le terminal de windows ne connaît pas ces commandes. Pour les faire fonctionner, il faut donc bien veiller à installer Qt dans le répertoire donné par défaut (généralement C:\Qt) et soit utiliser le terminal fourni avec Qt ou alors ajouter le répertoire bin de Qt au PATH, ce qui n'est pas fait automatiquement. Le principal problème de Windows, c'est (entre autres) que toutes les commandes ne sont pas centralisées sur un seul et même terminal, et certains logiciels ont leur propre terminal (c'est le cas de Qt et de git par exemple), ce qui ne simplifie pas les choses. Après avoir appris à maîtriser l'environnement Windows j'ai enfin pu installer les librairies nécessaires à la compilation du logiciel.

L'étape suivante consiste à générer les fichiers makefiles du logiciel, en utilisant cmake. Pour cela, il faut d'abord créer un répertoire build et se placer dedans pour ensuite générer les makefiles. La commande utilisée pour réaliser cette dernière action est la suivante :

```
cmake -G "MinGW Makefiles" -DCMAKE_BUILD_TYPE=Release
-DCMAKE_PREFIX_PATH="C:/Program Files (x86)/GnuWin32";
"C:/Qwt-6.1.3" ..
```

L'option « -G » suivie de « "MinGW Makefiles" » permet de spécifier quel type de fichier makefiles doivent être générés, dans notre cas on souhaite des fichiers makefiles compatibles avec le compilateur MinGW ; La variable CMAKE_BUILD_TYPE indique le type de build obtenu (« Debug » ou « Release ») et la variable CMAKE_PREFIX_PATH permet d'indiquer l'emplacement des librairies nécessaires à la création des makefiles.

Après avoir fait cela, j'ai ensuite corrigé le guide de compilation afin de le rendre plus complet et plus détaillé.

2.1.2 Réalisation d'un script pour la compilation du logiciel

La compilation du logiciel nécessitant d'entrer plusieurs lignes de commande à la suite, elle était donc peu pratique. Il m'était ainsi donné comme objectif de simplifier ce processus d'installation en écrivant

un mode d'emploi pas à pas, afin d'expliquer la démarche à suivre pour les enseignants mais aussi les étudiants souhaitant compiler le logiciel.

J'ai dans un premier temps eu l'idée d'écrire un script perl afin que les lignes de commandes se lancent directement depuis le script. Ainsi l'utilisateur n'a plus qu'à appeler le script perl dans un terminal (ou le « CMD Qt 5.8 for Desktop (MinGW 5.3.0 32 bit) » sur Windows) afin que le processus d'installation du logiciel se déroule de manière automatique.

J'ai choisi le langage perl car c'est un langage de script, il est donc bien adapté pour ce type de fonctions. De plus ayant eu l'occasion de le pratiquer durant mon année universitaire, il était donc plus naturel pour moi de choisir ce langage plutôt qu'un autre langage de script tel que python par exemple.

Pour appeler des commandes dans un script perl, il y a plusieurs solutions :

La commande system (\$commande, @arguments) qui exécute ladite commande et continue le script après son exécution.

La commande exec (\$commande, @arguments) qui exécute la commande mais termine le script après son exécution (comme un return dans une fonction).

La commande qx/\$commande/ qui permet de récupérer l'« output » de la commande dans la sortie STDOUT.

Vu que je n'avais pas besoin de récupérer la sortie de la commande, la fonction qx ne m'était pas très utile. De plus comme il était nécessaire d'exécuter plusieurs commandes à la suite il n'était pas très approprié d'utiliser exec, c'est pourquoi j'ai choisi d'utiliser la fonction system pour appeler les commandes nécessaires à la compilation dans mon script perl.

Cependant la fonction exec m'a quand même été utile. En effet, après avoir réalisé la compilation à l'aide de mon script, j'ai souhaité compléter ce dernier pour qu'il lance le logiciel ImageINSA directement après avoir terminé de le compiler, toujours dans un souci de simplification de la démarche de lancement du logiciel. Néanmoins lorsque j'utilisais la fonction system pour réaliser cette action, le terminal se mettait en attente tant que le logiciel était ouvert, il n'était donc plus possible de s'en servir. Même s'il y a la possibilité d'ouvrir un autre terminal, je ne souhaitais pas bloquer le terminal déjà ouvert lorsque le logiciel est en cours d'exécution.

C'est alors que j'ai eu l'idée d'utiliser la fonction exec pour réaliser le lancement du logiciel. En effet, cette dernière, comme précisé plus haut, termine le script après son exécution, ce qui a pour conséquence que le terminal n'est plus bloqué lorsque le logiciel est lancé puisqu'au moment où celui-ci a été lancé le script s'est terminé grâce à la fonction exec. J'ai ainsi pu lancer le script tout en gardant le terminal actif.

Une autre difficulté liée à cette tâche était que le répertoire dans lequel se trouvent les sources du logiciel varie d'un utilisateur à l'autre. En effet, les sources se trouvent normalement dans le répertoire personnel de l'utilisateur, qui correspond à son identifiant. Il est donc différent pour chaque personne. Il fallait donc trouver une fonction permettant de trouver le répertoire personnel de l'utilisateur. J'ai pour cela utilisé la fonction File : :HomeDir->my_home qui retourne le répertoire personnel de l'utilisateur courant.

Le script perl mis en œuvre est disponible en annexe A.

J'ai également écrit un mode d'emploi pas à pas pour installer le logiciel sur la machine virtuelle (x86), depuis le clonage du projet git jusqu'à la compilation, afin de rendre ce processus plus simple pour les utilisateurs et développeurs futurs (voir annexe A).

2.2 Gestion du projet sous git

Ce stage a également été l'occasion pour moi de me perfectionner à la gestion de version par l'intermédiaire du logiciel Git ¹. En effet, je n'avais eu que peu d'occasions d'utiliser ce logiciel auparavant et ce stage fut une opportunité de l'utiliser sur un vrai projet. Les sources du logiciel étant sur github j'ai donc dû me familiariser avec les commandes git mais aussi avec la méthodologie à adopter lorsqu'on utilise ce type de logiciel (quand et à quelle fréquence faut-il faire des commits ou des push par exemple). De plus le projet github avait la particularité d'utiliser un sous-module git afin d'intégrer le sous-projet detiq-t au projet. J'ai donc eu l'occasion d'appréhender cette structure de projet quelque peu particulière et qui n'est pas facile à assimiler de prime abord car l'intégration de sous-modules à un projet git entraîne des subtilités qui obligent le développeur à être totalement conscient de ce qu'il fait car cela peut générer des problèmes de version du sous-module.

J'ai dans un premier temps fait un « fork » sur mon profil github du repository nommé eimage qui est le nom du repository associé au logiciel ImageINSA. J'ai ensuite créé une branche à partir de la branche créée par Antoine Pazat lors de son stage à l'été 2016 afin de pouvoir faire des commits de mes modifications dessus. Cependant lorsque je modifiais la librairie Detiq-t qui est gérée par l'intermédiaire

1. Git est un logiciel de gestion de versions décentralisé

d'un sous-module git nommé `detiq-t`, il m'était impossible de faire un « push » de mes modifications car je n'avais pas les droits d'accès au repository `detiq-t`. Si je ne réalisais pas de « push » sur le sous-module `detiq-t`, les modifications apportées à ce sous-module n'étaient pas enregistrées sur le dépôt distant. Il a donc fallu que je fasse également un « fork » de ce repository afin de pouvoir pousser mes modifications dessus. J'ai donc changé l'URL du submodule dans le fichier `.gitmodules` en le remplaçant par celui du « fork » de `detiq-t`. Ainsi, je pouvais pousser mes améliorations apportées à `detiq-t` et au logiciel ImageNSA sur le dépôt distant.

Cette expérience m'a appris à me méfier des sous-modules dans le futur car même s'ils peuvent paraître très pratiques de prime abord, ils induisent en réalité bien plus de contraintes et peuvent être source d'erreurs, particulièrement si plusieurs personnes travaillent sur un même projet et ne maîtrisent pas toutes leurs subtilités.

2.3 Extension du logiciel

Nous verrons dans cette parties les différentes fonctionnalités que j'ai ajoutées au logiciel durant mon stage. Même si certaines de ces fonctionnalités ont été rajoutées de ma propre initiative, la plupart d'entre-elles étaient en réalité demandées par les professeurs. Toutes les suggestions d'améliorations figuraient dans le fichier de log de l'évolution du logiciel ImageNSA[2].

2.3.1 Ajout des boutons « sauvegarder » et « zoom »

La fonction zoom était déjà mise en œuvre dans le logiciel mais elle ne l'était que sous la forme d'un raccourci clavier (touche `Ctrl` enfoncée + molette de la souris) et les utilisateurs ne connaissant pas le raccourci n'avaient aucun moyen d'utiliser cette fonctionnalité. Il avait ainsi été réclamé par plusieurs enseignants l'ajout d'une fonction zoom accessible en faisant un clic droit sur l'image. Le logiciel ImageNSA utilisant la bibliothèque `Detiq-t` pour la gestion (entre autres) de l'interface graphique grâce à l'interface générique, c'est donc sur celle-ci que j'ai réalisé les modifications nécessaires. J'ai ainsi modifié le fichier `imageWindow` qui a pour but de gérer toutes les fonctions relatives à la fenêtre dans laquelle s'affiche l'image.

L'ajout d'une action dans le menu contextuel (clic droit) d'une image se fait en deux étapes :

- Déclaration de l'action et ajout dans le menu contextuel
- Mise en œuvre de la méthode réalisant l'action

Par exemple pour la fonction « zoom + » la déclaration ressemble à ceci :

```
menu()->addAction(tr("Zoom +"), this, SLOT(zoom_in()));
```

Le champ `SLOT` représente la méthode appelée lorsque l'on clique sur le bouton « Zoom + ».

Pour mettre en œuvre la fonction `zoom_in()` (resp. `zoom_out()`) correspondant à un zoom (resp. dézoom), j'ai dû adapter la fonction existante `zoom()` réservée au zoom à l'aide du raccourcis clavier.

J'ai ensuite décidé d'ajouter cette fonction de zoom aux boutons situés juste en-dessous de l'image (appelée barre de statut), afin de la rendre encore plus accessible (voir figure 2.1).

L'ajout d'une fonctionnalité à la barre de statut se fait en trois temps :

- Déclaration du bouton que l'on souhaite ajouter dans le header et dans le fichier `cpp` ainsi que de ses caractéristiques
- Mise en œuvre de la méthode réalisant l'action et connexion du bouton à cette méthode à l'aide de la méthode `QObject : connect`
- Affichage du bouton (ajout du bouton dans la barre de statut)

Pour ajouter une icône à une fonctionnalité, nous voulions que l'icône soit inclus dans l'exécutable afin que les utilisateurs ne puissent pas la modifier. Pour cela, il faut au préalable ajouter l'image correspondant à l'icône dans le fichier `.qrc` afin de pouvoir la retrouver. On peut ensuite appeler l'image dans le code source en mettant « `:/` » devant le chemin de l'image.

J'ai également décidé d'ajouter la fonction « enregistrer sous » dans le menu contextuel ainsi que dans la barre de statut de l'image (voir figure 2.1). La méthode relative à cette action m'a demandé plus de temps pour la mettre en œuvre mais j'ai également trouvé une solution en adaptant la méthode `saveAs()` présente dans la barre de menu Fichier à ma situation, ce qui ne fut pas chose aisée.

Afin de rendre le travail des étudiants plus agréable notamment au moment de la rédaction de leur compte rendu, j'ai également ajouté la fonction « enregistrer sous » pour tous les histogrammes. Ils pourront donc dorénavant sauvegarder l'image de leur histogramme et l'importer dans un document word par exemple, là où auparavant ils devaient réaliser une capture d'écran puis la rogner ensuite. Toujours dans ce souci de facilitation, j'ai ajouté la fonction « copier image » qui leur permettra dorénavant comme son nom l'indique de faire un copier/coller de n'importe quelle image ou partie d'image.

2.3.2 Ajout bouton pour supprimer la normalisation des images de type double

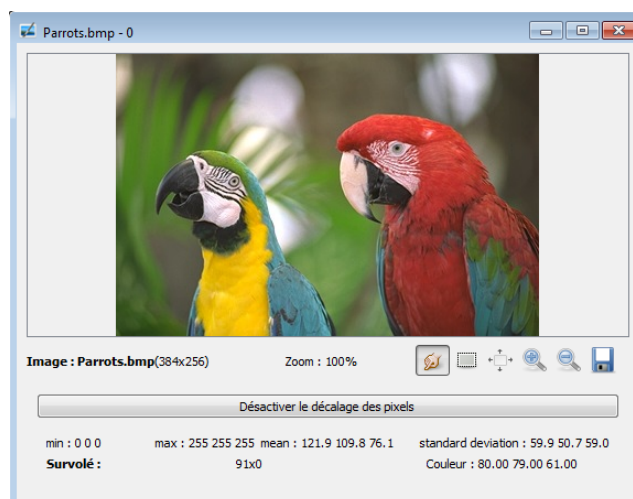


FIGURE 2.1 – Aperçu de la fenêtre image avec les boutons de zoom et de sauvegarde ainsi que le bouton « Désactiver le décalage des pixels » permettant de supprimer la normalisation

Un autre problème du logiciel était que les images ayant des valeurs de pixels négatives étaient normalisées pour ramener ces valeurs entre 0 et 255. Cependant cela posait un problème car les pixels ayant la valeur 0 et qui sont donc théoriquement noirs étaient ici affichés en gris car on avait changé leur valeur d’affichage de 0 à 127. Cette méthode a certains avantages comme le fait de pouvoir mieux distinguer les pixels à valeur négative de ceux à valeur positive. En effet, si on ne fait pas cette normalisation entre 0 et 255, les pixels négatifs auront la même couleur que les pixels à 0, c’est-à-dire noir. La normalisation permet donc de distinguer rapidement sur les images d’erreur par exemple, les valeurs fortement négatives, qui apparaissent de plus en plus foncées, les valeurs à 0 qui sont en gris et les valeurs positives qui sont de couleur claire.

Cependant cette normalisation présente un inconvénient pédagogique majeur. En effet, il y a un risque que les étudiants, lorsqu’ils analysent une image d’erreur sans regarder la grille de pixels, pensent que les pixels de couleur gris soient positifs, ce qui nuit à la compréhension des algorithmes.

L’idée était donc de rajouter une fonction permettant de supprimer ce décalage réalisé lors de l’affichage des pixels. J’ai donc décidé d’ajouter un bouton qui lorsqu’il est pressé, affiche dans une autre fenêtre l’image sans ce décalage, afin que les étudiants puissent voir l’affichage réel de l’image et mieux se rendre compte du traitement effectué. J’ai dans un premier temps voulu appliquer ce traitement directement sur l’image normalisée, c’est-à-dire en la modifiant, mais le fait d’ouvrir une nouvelle fenêtre pour afficher l’image sans décalage s’est avéré plus pertinent car cela permet d’avoir les deux images sous les yeux pour pouvoir les comparer et de continuer à travailler sur l’image originale ensuite car elle reste plus agréable pour distinguer les valeurs positives des valeurs négatives.

2.3.3 Résultats Double pour les opérations sur les images standards

Le logiciel ImageNSA propose de réaliser des opérations sur les pixels d’une image. Il est donc possible de modifier les valeurs d’une image soit en fonction d’une constante, soit en fonction des valeurs d’une autre image. Cependant, lorsque l’on souhaitait réaliser des opérations sur des images standards (c’est-à-dire ayant des valeurs entières et positives), le résultat était donné en image standard. Or il est parfois plus judicieux que le résultat soit donné en format Double, ce qui permettrait d’avoir des valeurs réelles et négatives. En effet si on réalise une soustraction d’une image par une autre, il y a des chances que certains pixels aient des valeurs négatives. Dans ce cas le logiciel remplaçait ces valeurs par 0.

J’ai donc décidé d’ajouter une case à cocher dans la boîte de dialogue « opérations sur les pixels » (voir figure 2.2 afin de pouvoir bénéficier d’un résultat en format Double même si on réalise une opération sur des images standards. J’ai pour cela modifié le fichier PointOp.cpp afin d’ajouter une « QCheckBox » à la fenêtre de dialogue. Si l’opération est effectuée sur une image au format Double, la case est obligatoirement grisée et cochée. J’ai ensuite fait en sorte que le booléen dblResult vaille « true » lorsque la case est cochée.

Ainsi il est donc dorénavant possible d’obtenir un résultat en format Double pour des opérations réalisées sur des images standards, ce qui pédagogiquement est plus approprié car cela permet de mieux comprendre les résultats obtenus lors des opérations sur les pixels.

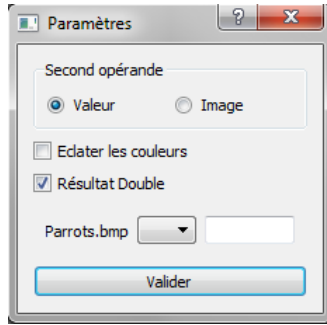


FIGURE 2.2 – Aperçu de la boîte de dialogue avec l’option « Résultat Double »

2.3.4 Correction de l’affichage de l’élément structurant

J’ai également remarqué que la création de l’élément structurant, permettant de réaliser des opérations telles que l’érosion ou encore l’ouverture, présentait quelques problèmes.

En effet, au moment de créer cet élément structurant, il est possible de définir un paramètre « échelle », en plus du paramètre de taille de l’élément structurant. Ce paramètre permet de multiplier la taille de l’élément structurant par la valeur de ce premier.

Cependant, si ce paramètre d’échelle semblait fonctionner à en juger des résultats obtenus en réalisant l’érosion d’une image simple avec un élément structurant dont le paramètre échelle était modifié, l’affichage de l’élément structurant au moment de sa création était incorrect car celui-ci ne s’adaptait pas à la valeur de l’échelle.

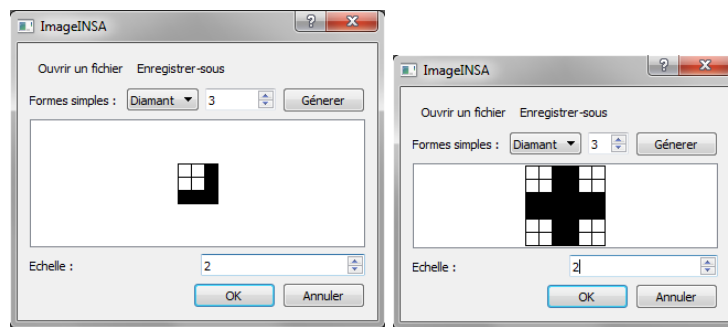


FIGURE 2.3 – Affichage de l’élément structurant avant et après correction

On s’aperçoit sur l’image de gauche qu’on ne voit qu’une petite partie de l’élément structurant, qui normalement représente une croix. J’ai donc corrigé cela en modifiant le code car celui-ci comportait des instructions empêchant la figure d’adapter son affichage à l’échelle.

Après correction (image de droite), on constate que l’on voit maintenant l’élément structurant dans sa totalité, et ce avec les mêmes paramètres que sur l’image précédente.

Ainsi la création de l’élément structurant est plus claire pour les étudiants.

J’ai également rendu possible le fait de « blanchir » une case noire en cliquant dessus. En effet, le logiciel proposait déjà de noircir les cases blanches par simple clic sur celles-ci cependant il était impossible de faire machine arrière pour les reblanchir si besoin il y avait.

2.3.5 Traduction français/anglais

Enfin, étant donné que l’utilisateur a la possibilité de choisir la langue du logiciel entre l’anglais et le français, il a fallu mettre en œuvre la traduction du terme « Save As ». Pour cela il a fallu ajouter dans le fichier .ts le terme que l’on veut traduire (ici «Save as») et la traduction en français (« Enregistrer sous »). Cela se présente sous cette forme :

```
<message>
  <source>Save As</source>
  <translation>Enregistrer sous</translation>
</message>
```

Ainsi lorsque l’on passe de l’anglais au français et vice-versa, le texte est automatiquement modifié. Cette étape de traduction a été nécessaire tout au long de mon stage à chaque fois que j’ajoutais du texte quelque part, afin d’assurer la disponibilité du logiciel dans les deux langues.

2.4 Correction des algorithmes

L'un des objectifs de ce stage était de corriger quelques erreurs présentes dans les algorithmes de traitement d'image. En effet, lors de l'utilisation en travaux pratiques, le logiciel comporte quelques incohérences que les différents professeurs utilisant le logiciel ont pu relever.

2.4.1 Inventaire des corrections à apporter

J'ai tout d'abord commencé par faire l'inventaire des bugs répertoriés afin de voir s'ils étaient présents ou non sur le logiciel. J'ai donc pour cela réalisé un tableau excel répertoriant les bugs ainsi que leur présence ou non dans la version du logiciel utilisée pour les TP mais aussi la version mise à jour par Antoine Pazat lors de son stage à l'été 2016 et qui n'a pas pu être testée. Cette partie m'a demandé plus de temps que prévu car étant novice en matière de traitement d'image, il était parfois compliqué pour moi de déterminer si le résultat d'un traitement réalisé sur une image était correct ou non, ce qui posait problème pour détecter les bugs.

2.4.2 Correction entropie

J'ai dans un premier temps réalisé que le calcul d'entropie était incorrect sur la dernière version du logiciel (celle qui a été réalisée par le stagiaire Antoine Pazat à l'été 2016) alors qu'il semblait correct sur la version de TP. Après vérification auprès de Luce Morin, il a été convenu que le bon calcul d'entropie était celui réalisé par la version de TP. J'ai donc corrigé ce calcul sur la dernière version du logiciel.

De plus le logiciel présentait l'inconvénient de ne pas pouvoir calculer l'entropie d'une image de type Double, c'est-à-dire ayant potentiellement des valeurs négatives. La résolution de ce problème était très complexe car ce calcul avait depuis le départ été pensé pour être réalisé sur des images de type standard, il a donc fallu que je réadapte la majorité des calculs pour que ceux-ci puissent s'effectuer sur une image de type Double. En effet, j'ai même été contraint de rajouter une classe « DoubleEntropyOp » dans les opérations du logiciel afin que le calcul puisse prendre en compte ce type d'images. Le calcul d'entropie tel qu'il était mis en œuvre ne pouvait prendre en paramètre que des images de type standard, car celui-ci héritait de la classe Operation, qui ne fonctionne qu'avec des images de ce type. J'ai donc fait hériter ma nouvelle classe « DoubleEntropyOp » de la classe DoubleOperation qui elle, comme son nom l'indique, fonctionne avec des images Double.

Cependant cela n'était pas suffisant car le calcul d'entropie repose sur l'histogramme de l'image. Lui aussi n'était à l'origine pas réservé aux images de type Double car il n'était pas fait pour prendre des valeurs négatives. L'histogramme d'une image représente le nombre de pixels en fonction de la valeur du pixel. A l'origine il n'allait que de 0 à 255 (pour une image standard), cependant les images de type Double peuvent prendre des valeurs négatives allant jusqu'à -255. J'ai donc dû modifier la classe « Histogram » pour que les valeurs des abscisses puissent fluctuer entre -255 et 255. Il fallait également initialiser le nombre de pixels à 0 pour chaque valeur de pixel possible afin que celles-ci ne soient pas initialisées aléatoirement, ce qui fausserait le calcul. Une fois ces changements mis en œuvre, il a fallu veiller à ce que le parcours de l'histogramme de l'image se fasse uniquement dans l'intervalle entre la plus petite valeur de pixel et la plus grande valeur de pixel dans l'image. En effet, si la plus petite valeur de pixel est -157, il n'est pas nécessaire d'avoir à parcourir l'histogramme entre -255 et -158 car on sait que toutes ces valeurs seront à zéro (même chose pour la valeur maximale de pixel). Cela permet de réduire quelque peu le temps de calcul.

Une fois toutes ces considérations prises en compte, le calcul d'entropie pour les images de type Double est devenu possible. Cette amélioration fut une des plus complexes à mettre en œuvre car elle nécessitait de réellement bien maîtriser l'architecture du logiciel afin dans un premier temps de comprendre pourquoi ce calcul était impossible sur les images de type Double et dans un second temps d'apporter les modifications nécessaires pour que le calcul fonctionne sur ce type d'image.

2.4.3 Correction transformée de Hough

La transformée de Hough proposée par le logiciel nécessitait plusieurs améliorations et corrections. Tout d'abord, la transformée de Hough est une transformée permettant de détecter les lignes dans une image. Chaque ligne d'une image est représentée par un point dans l'espace de Hough. Dans le cours de traitement d'image réservé aux étudiants du département EII, le repère de l'image est présenté avec les y croissants vers le haut alors que sur le logiciel c'est l'inverse, les y sont croissants vers le bas (cf. illustration). Il a donc fallu réaliser un changement de repère afin que les calculs mais aussi l'affichage des résultats soient cohérents avec le photocopié de cours des étudiants.

J'ai également décidé de remplacer les valeurs de pixels survolées et sélectionnées par leurs coordonnées dans le domaine de Hough (avec ρ et θ), afin que la lecture des résultats soit plus claire pour les étudiants. Par exemple, le pixel (0,0) affiche maintenant (ρ : 0, θ : 180). Pour faire cela, il a d'abord fallu que je vérifie que l'opération effectuée est bien une transformée de Hough. J'ai pu faire cela

dans la méthode « outDoubleImage » du fichier Operation.cpp en testant si l'opération en cours est une transformée de Hough grâce à cette ligne de code : `if(HoughOp* v = dynamic_cast<HoughOp*>(this))` Si ce test est vrai, cela appelle la méthode isHough de la classe DoubleImageWindow qui passe le booléen `_hough` de cette classe à vrai, et donc change la façon dont les coordonnées seront affichées (dans ce cas, elle seront affichées comme indiqué ci-dessus).

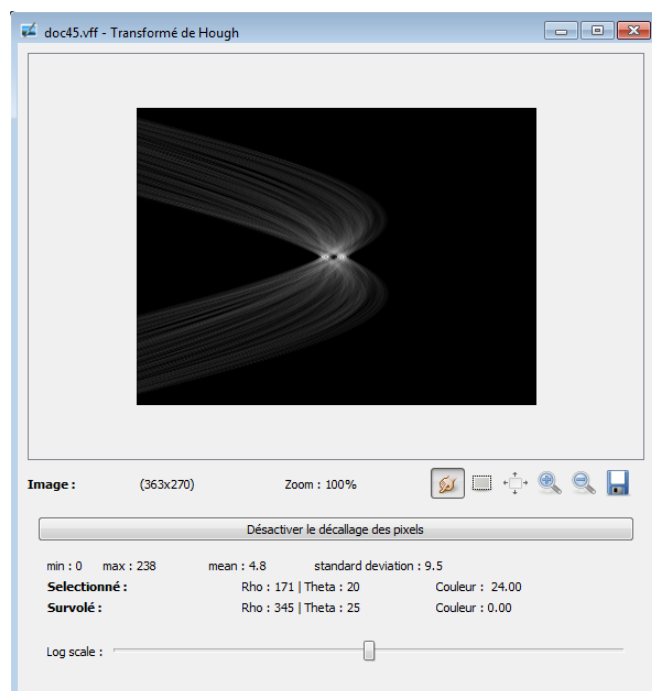


FIGURE 2.4 – Exemple de transformée de Hough

De plus, pour ce qui est de la méthode n°2 de calcul de la transformée de Hough, il est possible de définir un pas pour les angles, ainsi que pour les distances. L'affichage de la transformée de Hough et plus particulièrement des valeurs de pixel survolés et sélectionnés dépend donc de ces valeurs. Il a donc fallu que je les passe en paramètres afin de pouvoir en tenir compte au moment du calcul de la valeur affichée pour Rhô et Thêta car je ne pouvais pas avoir accès à ces variables depuis la classe DoubleImageWindow. J'ai donc ajouté les attributs `angleStep` et `distanceStep` à la classe DoubleImageWindow et modifié la méthode `isHough` pour qu'elle écrive les valeurs dans ces attributs afin de pouvoir les réutiliser.

2.4.4 Ajout profil ligne/profil colonne

Je me suis ensuite chargé de mettre en œuvre la fonctionnalité permettant de faire un profil ligne ou un profil colonne sur une image. Le profil ligne (resp. colonne) permet d'afficher l'histogramme des valeurs des pixels de l'image sur une ligne (resp. colonne) donnée.

La fonction semblait être déjà présente dans le logiciel dans la mesure où les boutons « profil ligne » et « profil colonne » étaient déjà présents dans le menu contextuel de l'image, mais le traitement réalisé par ces boutons ne correspondait pas du tout à ce qui était attendu. En effet, aucune ligne ou colonne n'étaient demandée à l'utilisateur et l'histogramme affiché était le même pour toutes les images et présentait des valeurs aberrantes. Le traitement réalisé par ces boutons était donc inexistant, j'ai donc dû le mettre en œuvre.

Ainsi, j'ai dans un premier temps créé deux fichiers template dans la bibliothèque imagein que j'ai appelés « LineProfile.tpp » et « ColumnProfile.tpp » (accompagnés de headers). Ces fichiers ont pour but de faire le calcul des données qui serviront ensuite à créer le profil correspondant.

Ce traitement consiste à parcourir tous les pixels de la ligne (resp. colonne) passée en paramètre et à remplir un tableau dont les indices représentent les numéros des colonnes (resp.lignes) et les valeurs correspondent pour chaque indice à la valeur prise par le pixel aux coordonnées de la ligne et de la colonne. Ceci permet de construire les données du profil.

J'ai également créé des fichiers nommés « RowWindow », « RowView », « ColumnWindow » et « ColumnView » afin de gérer l'affichage des profils. RowWindow et ColumnWindow héritent de GenericHistogramWindow tandis que RowView et ColumnView héritent de GenericHistogramView.

Ces fichiers permettent de spécifier des informations qui ne sont propres qu'aux profils ligne et profils colonne tels que le titre du graphique, ou encore le titre et l'échelle des axes.

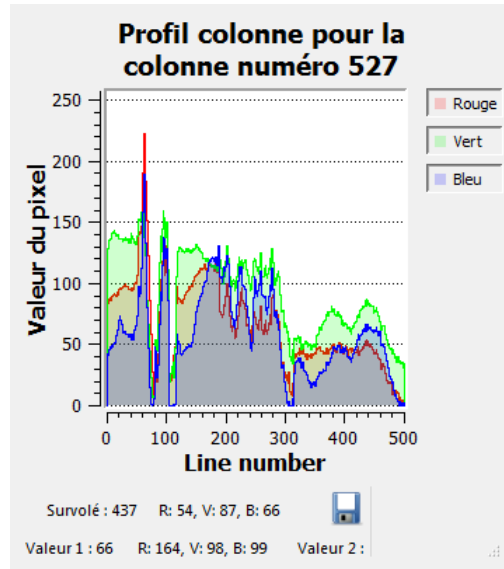


FIGURE 2.5 – Exemple de profil colonne

L'utilisateur doit également spécifier la ligne ou la colonne pour laquelle il souhaite établir le profil. Pour cela j'ai utilisé la classe `QInputDialog` qui permet de créer assez facilement une fenêtre de dialogue permettant de récupérer une variable.

L'objectif était que l'étudiant puisse choisir la ligne ou la colonne dont il souhaite établir le profil en cliquant directement sur l'image. Pour répondre à cela, j'ai eu l'idée de mettre comme valeur par défaut dans la boîte de dialogue le numéro de la ligne ou de la colonne (suivant le type de profil) correspondant au pixel sélectionné par l'utilisateur. L'utilisateur peut bien évidemment modifier cette valeur dans la boîte de dialogue, soit en tapant directement un numéro, ou bien en ajustant la valeur par défaut avec les flèches.

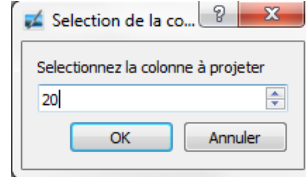


FIGURE 2.6 – Boîte de dialogue permettant de choisir la ligne/colonne à projeter

2.4.5 Correction projection horizontale/verticale

Il a été convenu que l'histogramme de projection horizontale (resp. verticale) représenterait le nombre de pixels à 255 pour chaque ligne (resp. colonne) de l'image pour une image binaire et le nombre de pixels supérieurs ou égaux à une valeur seuil pour chaque ligne (resp. colonne) de l'image pour une image en niveaux de gris.

Cette fonctionnalité semblait présente sur le logiciel mais ne fournissait pas du tout le résultat escompté. J'ai donc dû réécrire l'algorithme permettant de générer les données nécessaires à la construction de l'histogramme dans le fichier `ProjectionHistogram.tpp` afin qu'il réalise la tâche précisée ci-dessus. Pour ce qui est de la boîte de dialogue permettant à l'utilisateur d'entrer la valeur seuil, la valeur par défaut correspond à la valeur du pixel sélectionné par l'utilisateur sur l'image.

Ainsi, pour une image binaire, peu importe la valeur du seuil, si celui-ci est supérieur à zéro, le résultat affiché sera le nombre de pixels à 255 pour chaque ligne ou colonne (selon le type de projection choisi) puisqu'une image binaire n'a que des pixels à 0 ou à 255. Pour ce qui est d'une image en niveau de gris, le résultat affiché sera le nombre de pixels supérieurs ou égaux à la valeur seuil pour chaque ligne ou colonne. Enfin, pour une image couleur, le logiciel affichera pour chaque couleur le nombre de pixels supérieurs ou égaux à la valeur seuil pour chaque ligne ou chaque colonne.

Conclusion

En conclusion, je peux dire que le travail que j'ai réalisé durant ce stage a permis d'aboutir à une version plus complète du logiciel ImageINSA, qui intègre les améliorations faites pendant mon stage ainsi que celles faites par A. Pazat[4], avec certaines fonctionnalités ayant été ajoutées et d'autres ayant été corrigées, mais aussi plus ergonomique, bien que la version précédente disposait déjà d'une ergonomie correcte. Ainsi cette nouvelle version sera probablement utilisée pour les travaux pratiques des étudiants de quatrième année EII. De plus le processus de compilation a été simplifié ce qui devrait faciliter l'extension future du logiciel. Il reste en effet des améliorations à apporter au logiciel. Celles-ci ont été recensées dans un tableau Excel que j'ai réalisé (voir Annexes D et E). Il faudra également tester le logiciel sur la machine virtuelle 64 bits et vérifier que la compilation fonctionne sous Linux.

D'un point de vue personnel, j'ai trouvé très intéressant en tant qu'étudiant de travailler sur un logiciel à but pédagogique, car cela m'a probablement permis d'avoir un point de vue différent de celui d'un ingénieur, et j'ai pu apporter mon ressenti concernant certains aspects du logiciel. De plus, ce stage m'a permis d'apprendre à travailler de manière autonome sur un projet d'assez grande envergure. Cela m'a notamment donné l'occasion d'apprendre à utiliser le logiciel git de manière plus poussée particulièrement à travers la gestion des sous-modules. Cela me sera forcément très utile en tant qu'ingénieur car git est aujourd'hui un outil très répandu et il est primordial de savoir s'en servir. Cela m'a permis d'apprendre qu'il faut éviter l'utilisation des sous-modules git car cela peut être source d'erreurs, notamment lorsqu'il y a plusieurs personnes à travailler sur un même projet.

J'ai aussi eu l'occasion de découvrir plus en profondeur la langage C++ que je n'avais pas ou peu utilisé jusqu'à présent. J'ai aussi eu l'opportunité de manipuler le framework Qt, qui est très pratique lorsqu'il s'agit de réaliser des interfaces graphiques, même si c'est loin d'être sa seule utilité, il me sera donc très certainement utile dans le futur.

Ces huit semaines m'ont également permis d'acquérir quelques notions sur le traitement d'image, domaine dans lequel j'étais novice. Cela fut très enrichissant non seulement pour ma culture personnelle, mais aussi car cela m'a permis de faire de l'informatique dans un domaine jusque-là inconnu pour moi, ce qui je pense est une des spécificités du métier d'ingénieur.

Enfin, ce stage m'a permis de découvrir le quotidien d'un laboratoire de recherche pendant huit semaines, ce qui fut très intéressant pour ma part afin d'avoir une idée un peu plus précise de ce que représente le métier de chercheur.

D'une manière générale, j'ai beaucoup apprécié faire ce stage et travailler sur le logiciel ImageINSA et je trouve que ces huit semaines ont été très enrichissantes et formatrices pour moi.

Bibliographie

- [1] Dépôt github du logiciel imageinsa. www.github.com/eiimage/eiimage.
- [2] Update imageinsa. Fichier log de l'évolution du logiciel ImageINSA, oct 2012.
- [3] Benoit Averty, Samuel Babin, Matthieu Bergère, Thomas Letan, Sacha Percot-Tétu, and Florian Teyssier. Projet detiq-t. Rapport final, may 2012.
- [4] Antoine Pazat. Extension d'un logiciel pédagogique de traitement d'images. Rapport de stage, aou 2016.

Annexes

Annexe A

Script pour la compilation du logiciel ImageINSA

```
use File::HomeDir;

#Récupère le répertoire de l'utilisateur
chdir(File::HomeDir->my_home);

chdir('ImageINSA_dev\\imageinsa');

#Création du répertoire build
system('mkdir build');

#Se place dans le répertoire du build
chdir('build');

system("cmake -G \"MinGW Makefiles\" -DCMAKE_BUILD_TYPE=Release
-DCMAKE_PREFIX_PATH=\"C:/Program Files (x86)/GnuWin32\";\"C:/Qwt-6.1.3\" ..");

system("mingw32-make -j4");

chdir(File::HomeDir->my_home);

#Lancement du logiciel ImageINSA
exec('ImageINSA_dev\\imageinsa\\build\\ImageINSA\\imageinsa.exe');
```

Annexe B

Mode d'emploi pour la compilation du logiciel sur la machine virtuelle (x86)

Compilation sur testingAdmin :

Ouvrir session testingAdmin

Créer dossier ImageINSA_testCompilation dans le dossier testingAdmin

Clic droit -> git bash here dans le dossier ImageINSA_testCompilation

Dans la fenêtre de git Bash :

```
git clone https://github.com/qbigot/eiimage.git
```

```
cd eiimage
```

```
git checkout qb2017
```

```
git submodule update --init
```

```
cd lib/detiqt
```

```
git checkout qb2017
```

Ouvrir terminal Qt :

```
cd c:\Users\testingAdmin\ImageINSA_testCompilation\eiimage
```

```
mkdir build
```

```
cd build
```

```
cmake -G "MinGW Makefiles" -DCMAKE_BUILD_TYPE=Release  
-DCMAKE_PREFIX_PATH="C:/Program Files/GnuWin32";C:\Qwt-6.1.3 ..
```

```
mingw32-make -j4
```

Compilation sur testing :

Ouvrir session testing

Créer dossier ImageINSA_testCompilation dans le dossier testing

Clic droit -> git bash here dans le dossier ImageINSA_testCompilation

Dans la fenêtre de git Bash :

```
git clone https://github.com/qbigot/eiimage.git
```

```
cd eiimage
```

```
git checkout qb2017
```

```
git submodule update --init
```

```
cd lib/detiqt
```

```
git checkout qb2017
```

Ouvrir terminal Qt :

```
cd c:\Users\testing\ImageINSA_testCompilation\eiimage

mkdir build

cd build

//on ajoute le repertoire de mingw53_32 au PATH
//pour pouvoir utiliser les commandes qu'il contient pour la compilation
set PATH=%PATH%;C:\Qt\5.9.1\mingw53_32\bin

cmake -G "MinGW Makefiles" -DCMAKE_BUILD_TYPE=Release
-DCMAKE_PREFIX_PATH="C:/Program Files/GnuWin32";C:\Qwt-6.1.3 ..

mingw32-make -j4
```

Annexe C

Planning

Semaine du 12 juin 2017 au 16 juin 2017 : Prise en main de la compilation du logiciel, installation de l'environnement et prise en main de git.

Semaine du 19 juin 2017 au 23 juin 2017 : Prise en main de l'architecture du logiciel et inventaire des bugs présents dans le logiciel ainsi que des améliorations à apporter

Semaine du 26 juin 2017 au 30 juin 2017 : Mise en œuvre de quelques améliorations graphiques

Semaine du 3 juillet 2017 au 7 juillet 2017 : Passage en revue des différents éléments techniques de traitement d'image nécessaires pour mieux comprendre les traitements et les algorithmes et ainsi pouvoir apporter les modifications nécessaires au logiciel

Semaine du 10 juillet 2017 au 13 juillet 2017 : Finalisation de tous les détails techniques concernant la gestion du projet sous git, savoir la méthodologie à adopter pour pousser les modifications sur le dépôt distant et réalisation de tests de compilation du logiciel sur les différentes machines virtuelles

Semaine du 17 juillet 2017 au 21 juillet 2017 : Développement de plusieurs fonctionnalités pour améliorer le logiciel, correction des algorithmes

Semaine du 24 juillet 2017 au 28 juillet 2017 : Suite du développement des fonctionnalités pour améliorer le logiciel ; correction des algorithmes et finalisation des différents éléments à livrer (dépôt git, tableau récapitulatif des bugs, éventuellement le rapport, ...) avant départ en vacances.

Semaine du 28 août 2017 au 1^{er} septembre 2017 : Finalisation du rapport et de tous les éléments à livrer et correction de certains bugs

Annexe D

Tableau récapitulatif des bugs restants

Bug	Version qb2017 (modifications apportées)	Améliorations possibles
Huffman ne tient pas compte des pixels de valeur 0 =5 sur les images binaires, cela ne marche pas ! (utilise ancienne version histogramme). Serait intéressant de pouvoir l'appliquer sur des images d'erreurs de prédiction, ou sur coefficients transformés après quantification.	corrigé	VALIDE / NE MARCHE PAS SUR IMAGE de type DOUBLE / les valeurs dans la fenetre log pour P[xx] ne correspondent pas aux valeurs des pixels
Entropie : ne fonctionne que sur des images en niveau de gris. Là aussi intéressant de l'appliquer sur des images d'erreurs et coef. transformés.	correction entropie	VALIDE : indiquer dans la fenetre de log que l'entropie est la somme des entropies des 3 images R,V,B MARCHE SUR IMAGE DOUBLE AVEC VALEUR CORRECTE / indiquer dans la fenetre de log le pas de quantification utilisé
filtrage/Editer un filtre : pas possible de créer un filtre manuellement lors de l'exécution du logiciel en salle de TP (car problème de droits lors de la sauvegarde du masque dans un fichier?). mettre message d'erreur + possibilité de choisir le répertoire de sauvegarde du filtre)	Les filtres s'écrivent dans le fichier filters.xml situé dans le répertoire ImageNSA du build si interdit en écriture, télécharger le logiciel dans un répertoire personnel de l'étudiant	attention à conserver l'apparition du nouveau filtre dans le menu qui propose les différents filtres disponibles, même si on l'a sauvegardé dans un autre répertoire ; donner le choix du répertoire de sauvegarde du filtre créé et envoyer un message d'erreur si on n'a pas les droits en écriture
filtrage/Editer un filtre : le résultat de certains filtres prédéfinis ou édités est faux typiquement les filtres laplaciens dont les coefficients sont négatifs). existence d'une normalisation sur les filtres édités	La normalisation n'existe que lorsque l'on veut le résultat en Uchar, ce qui est normal vu qu'un Uchar ne peut pas être négatif	Revoir cela (peut-être à moi de voir avec Véro jusqu'à convergence des avis) / bien clarifier les vocabulaire (normalisation/offset) et ce qu'on veut faire dans le calcul / ou uniquement pour l'affichage de l'image résultat./ actuellement les coefficients du filtre sont normalisés implicitement (par la somme des coefficients si elle est non nulle) : il faudrait rendre cela explicite/ à fixer : quelles sont les options cochées par défaut et les valeurs par défaut (127 ou 128?) / pour moi les 2 options UChar et Double doivent faire le même calcul, mais dans le cas de UChar, on arrondit au plus proche entier et on crope entre 0 et 255 / faut-il offrir la possibilité d'appliquer un offset sur les valeurs en sortie du calcul ? pour moi il ne faut pas le faire de façon cachée / sur les filtres Roberts Sobel Prewitt, afficher les 2 images obtenues par chacun des filtres en plus de l'image de la norme du gradient et indiquer dans la fenetre de log comment est calculée la norme du gradient (racine des somme des carrés, ou somme des valeurs absolues) /Affichage du résultat : bouton désactiver l'offset : remplacer par un bouton qu'on peut
Hough 1 : on ne voit pas le bas de l'image résultat dernière ligne	?	

Hough 1 et Hough 2 : les angles varient dans un intervalle de taille 180° au lieu de 360° : revoir le calcul pour avoir rho positif et theta sur un intervalle de 360° en prenant rho et theta tels que définis dans le cours	correction transformée de Hough 2 (marche pour un angle step de 1; problème d'affichage de la valeur de theta si on met un autre angle step)====>réglé + prise en compte de distance step. Modification de HoughDialog.ui pour mettre la valeur par défaut de anglestep à 1	
Projection horizontale/verticale: on compte le nombre de pixels à 255 sur chaque ligne (la colonne) dans une image binaire. (histogramme donne le nbre de pixels en fonction du numéro de ligne). Pour une image en niveaux de gris, le sens de cette projection serait à préciser (on compte uniquement les pixels de valeur supérieur à un seuil fourni par l'utilisateur)	corrigé : l'histogramme affiche le nombre de pixels sur chaque ligne/colonne ayant une valeur supérieure à celle entrée par l'utilisateur (réglée par défaut sur la valeur du pixel sélectionné). L'orientation de l'histogramme doit-elle être inversée dans le cas de la projection horizontale (nombre de pixels en abscisse et numéro de ligne en ordonnée)?	VALIDE : affichage : mieux vaut rester cohérent avec ce qu'on fait pour les profils ligne/colonne
Profil ligne/colonne: devrait fournir les valeurs de niveau de gris (sous de courbe) sur la ligne (ou la colonne) dont on saisit le numéro (le profil ligne et profil colonne) + les rajouter dans le menu selectionne ligne/colonne avec la souris ou saisir un numero (afficher ligne selectionnee comme valeur par default + tracer la ligne sur l'image	corrigé : fichiers ColumnProfile et LineProfile créés dans imagein + ColumnView et ColumnWindow dans generic interface	VALIDE : mettre la même légende (numéro de ligne) que pour les histogrammes de projection / ca serait cool de pouvoir choisir la ligne interactivement sur l'image et/ou de visualiser la ligne choisie (1 trait sur la ligne ou 2 traits encadrant la ligne)
Element structurant : paramètre échelle pas clair : à clarifier : échelle prise en compte si on utilise tout de suite l'element structurant. Par contre l'échelle n'est pas prise en compte lors de la sauvegarde dans un fichier. Et la visualisation ne tient pas compte de l'échelle / possibilité de noirir les pixels mais pas d'annuler passer en mode switch	corrigé : ajout possibilité de blanchir les pixels noirs + résolu problème de l'affichage de l'élément structurant lorsqu'on change l'échelle Le parametre echelle ne peut pas être enregistré, l'élément structurant est enregistré à l'échelle 1 et l'utilisateur peut remodifier l'échelle après avoir chargé l'élément structurant	VALIDE : amélioration possible : pouvoir modifier pixel par pixel après avoir modifié l'échelle et sauvegarder l'élément structurant ainsi créé
DPCM : prédicteurs C et A+C/2 inversés (si on sélectionne l'un, cela lance l'autre)	modifié dans le fichier DPCMDialog.cpp	VALIDE : reste traduction en anglais à faire dans le fenetre log / à voir calcul de l'erreur de prédiction /
DPCM : test sur A,B,C pour le predicteur de Graham fait dans image originale et pas dans l'image reconstruite	corrigé dans DPCM.cpp	pas facile à vérifier!

Annexe E

Tableau récapitulatif des améliorations restantes

Améliorations	Version qb2017 (modifications apportées)	Améliorations possibles
Entropie : Pas de possibilité pour calculer l'entropie d'une image avec des valeurs négatives (ImageDouble) : ce serait utile pour les images d'erreur de prediction dont on veut évaluer le débit	Corrigé ! Création fichier DoubleEntropyOp pour le calcul d'entropie sur les images doubles (ps : une modification a dû être faite dans le fichier histogram.tpp : la déclaration du premier constructeur est devenue imagein::Histogram::Histogram(const imagein::Image_t<D>& img, unsigned int channel, const imagein::Rectangle& rect) : imagein::Array<unsigned int> (511) au lieu de 1 << (sizeof(D)*8) que j'ai laissé pour les autres constructeurs)	
Implémenter un quantificateur optimal (Lloyd-Max) : voir pourquoi plante avec 2 niveaux. Faire afficher la loi de quantif pour toutes les quantif. Remettre les quantif non lineaires dans le menu. Tester que les 2 propriétés du quantificateur optimal sont bien vérifiées		Fait planter le logiciel
Ajouter le zoom dans les menus (contextuel et barre de menu) en indiquant le raccourci (CTR-molette) + ajouter aussi par CTRL+ CTRL-	amélioré : ajout boutons zoom+ et zoom- dans la fenêtre et le menu contextuel de l'image	
Pouvoir faire une décomposition d'une image couleur en plans H,S,V / vérifier RVB->HSV->RVB redonne la même image :	En faisant une différence entre l'image originale et l'image reconstruite on remarque que tous les pixels ne sont pas à 0 (certaines valeurs sont à 1, peut-être à cause des arrondis)	Acceptable : à voir si possible faire du réversible exact en terme d'amélioration
Disposer d'un affichage logarithmique sur tout type d'image (pas seulement sortie de la DFT)	ajout de l'affichage logarithmique en sortie du DPCM	faire le log sur la valeur absolue AVANT de faire l'offset à 128 pour avoir un effet visuel utile
Améliorations seuillage: pouvoir modifier interactivement le 2ieme seuil comme le premier dans le cas du double seuil	il est possible de modifier le deuxième seuil avec le clic droit (indication ajoutée dans la fenêtre du seuillage)	VALIDE
Hough: revoir les repères pour avoir des angles trigonométriques (positifs dans le sens anti-horaire) et pour avoir les mêmes repères pour les résultats de Hough1 et Hough2 e les mêmes définitions que le cours	fait pour hough2	VALIDE pour Hough2 et Hough 1
DPCM : modifier l'ergonomie de la création de loi (boutons) car pas naturelle	non nécessaire	OK

DPCM : faire afficher l'image d'erreur des valeurs de prédiction avant quantification (pour pouvoir en mesure l'entropie et voir qu'elle a diminué)	ajout de l'image d'erreur en plus de l'image d'erreur de prediction + ajout de l'image de prediction + ajout de l'image d'erreur de codage	VALIDE / traduction en anglais dans la fenetre de log
DPCM : calculer et afficher l'entropie de l'image des vbaleurs de prédiction	voir si on peut supprimer la fonction codec	si on utilise la loi de quantif à 2 niveaux, c'est normal d'avoir un entropie proche de 1 / PB les valeurs de l'entropie indiquées dans la fenetre de log ne sont pas les mêmes que si on les calcule sur les images à partir du menu tools ; reprendre le code de DoubleEntropyOp pour voir si on obtient le même résultat
LUT/Pseudo-couleurs : pouvoir choisir la loi appliquée : nombre de couleurs, voire courbes R,V,B au choix		faire saisir nhue à l'utilisateur avec comme texte "nombre couleurs" "nb colors"
Filtrage/Display d'images réelles (ou toutes images) : ajouter des options de display : normalisation (valeurs affichées entre noir et blanc), off-set du blanc à la valeur 127, loi logarithmique,	ajouter bouton pour réaliser l'offset à 127 ("offset"/"decallage") sur les images doubles, juste pour l'affichage. La valeur dans la pixel grid doit rester inchangée //	bouton offset avec valeur par défaut 128 / bouton offset sur modèle du log-scale (bouton coché ou non et valeur 128 éditable)
Pouvoir sauvegarder l'img de l'histogramme, pouvoir sauvegarder les valeurs de l'histogramme dans un fichier	ajout possibilité de sauvegarder l'histogramme sous forme d'image	reste à faire : sauvegarde des valeurs de l'histogramme dans un fichier (et lecture?)
Opération sur les pixels : Pouvoir faire des différences d'img avec résultat en floatant (case à cocher dans les opérations « résultat réel », comme dans le cas des filtres)	lorsque l'on fait une opération sur les pixels avec une image ayant des nombres flottants, le résultat est donné en nombre flottant. Mettre un bouton "type de valeur du résultat" pour opérations sur les images standard	pour les opérations arithmétiques (+,-,*,/), reprendre les 2 options(Uchar,Double) comme pour les filtres / mettre la valeur Double par défaut si une des images d'entrée est Double/ si l'option Uchar est cochée, le résultat est arrondi au plus proche entier et croppé entre 0 et 255 (comportement actuel).
ajout bouton save as dans la fenêtre et le menu contextuel de l'image	fait	OK
ajout boutons zoom+ et zoom- dans la fenêtre et le menu contextuel de l'image	fait	OK
ajout fonction copier image permettant de copier l'image entière ou la sélection pour la coller ensuite. La fonction apparait dans le menu contextuel de l'image et dans le menu fichier et dans la barre de menu (barre avec les icones en-dessous des barres fichier, affichage, image, ...) ainsi que via le raccourci ctrl+c	fait	
Hough Inverse : passer la hauteur et la largeur de l'image en parametre	corrigé	

Autres bugs
Menu DMM : si on applique certaines opérations (ouverture, DMM,...à vérifier) avec un élément structurant non symétrique, il n'y a pas transposition de l'élément, et le résultat est donc erroné/ meme problème (mais pas le meme décalage..) avec le menu standard...;
Décomposition pyramidale : filtre QMF plante (parfois)

Autres améliorations
Filtrage : afficher aussi images Gx (gradient horizontal) et Gy (gradient vertical) pour les filtres Sobel/roberts/Prewit et renommer image résultat "norme du gradient" + indiquer le calcul dans la fenetre de log
Histogramme : pour des images en double, indiquer dans la fenetre log ou sur l'interface quel pas d'échantillonnage a été pris pour calculer l'histogramme
Convert to RGB image : créé une image vide, objectif à préciser / ne semble pas fonctionner sur une image double (plantage)
Menu contextuel : Revoir l'agencement des menus (regrouper histogramme et histogramme cumulé / faire une seule fonction Rogner avec action du copier-rognier actuel
Menu Image : faire une ligne de séparation entre « combine SVH planes » / « scaling »
Menu Show : revoir le choix des opérations à mettre dans ce menu
Plus globalement , revoir la répartition des fonctions dans les menus et les regroupements
Rendre réutilisable certains algos courants (calcul entropie) afin de ne pas re-coder n fois le calcul dans le logiciel
Proposer des conversions double vers uchar avec plusieurs choix à cocher ou pas : clipping/normalisation entre 0 et 255/0 placé à 128)
Possibilité de lecture d'un fichier jpeg2000

Table des figures

1	Capture d'écran du logiciel ImageINSA	4
1.1	Architecture du logiciel ImageINSA	5
2.1	Aperçu de la fenêtre image	10
2.2	Aperçu de la boîte de dialogue avec l'option « Résultat Double »	11
2.3	Affichage de l'élément structurant avant et après correction	11
2.4	Exemple de transformée de Hough	13
2.5	Exemple de profil colonne	14
2.6	Boîte de dialogue permettant de choisir la ligne/colonne à projeter	14

Résumé

Ce stage d'été de 3^e année a été réalisé au département EII de l'INSA de Rennes du 12 juin 2017 au 28 juillet 2017 puis du 28 août 2017 au 1^{er} septembre 2017. Ce stage porte sur l'extension du logiciel ImageNSA, qui est un logiciel de traitement d'images. Durant ce stage, après avoir réalisé une phase d'analyse du logiciel, plusieurs améliorations y ont été apportées concernant son ergonomie, plusieurs bugs ont également été corrigés et de nouvelles fonctionnalités ont été mises en œuvre pour ce logiciel. Ce rapport mentionne également la gestion du projet ImageNSA avec le logiciel git ainsi que la compilation du logiciel et l'installation de l'environnement logiciel nécessaire à son fonctionnement. Les différents outils utilisés durant ce stage étaient :

- le langage C++
- l'API Qt
- la bibliothèque annexe à Qt nommée qwt
- les librairies Zlib, RandomLib, Jpeg et LibPng
- le logiciel de gestion de version Git

Abstract

This 3rd year internship took place at the EII department of the INSA Rennes from June 12th 2017 to July 28th 2017 and from August 28th 2017 to September 1st 2017. This internship deals with the improvement of ImageNSA, which is an image processing software developed at the INSA. During this internship, after an analysis phase of the software, several improvements were made concerning the software ergonomics, many bugs were fixed and new features were added to it. This report also mentions the project management using Git as well as the compilation of the software and the installation of the software environment necessary to its functioning. The different tools used during this internship were :

- C++ language
- Qt API
- Qt additional library named qwt
- Zlib, RandomLib, Jpeg and LibPng libraries
- the version control system called Git