ENSG Géomatique
ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

Ecole Nationale des Sciences Géographiques

UNIVERSITÉ PARIS-EST

Université Paris-Est

-PROJET INFORMATIQUE -
Rapport de programmation

Mastère spécialisé ® Photogrammétrie, Positionnement, Mesure de Déformations

Développement d'une interface graphique pour améliorer des classifications de manière interactive

ENSG Géomatique
ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

CHUPIN Clémence

Novembre-Décembre 2017

☒ Non confidentiel ☐ Confidentiel IGN ☐ Confidentiel Industrie ☐ Jusqu'au ...

Ecole Nationale des Sciences Géographiques
6-8 Avenue Blaise Pascal - Cité Descartes - 77420 Champs-sur-Marne
Téléphone 01 64 15 31 00 Télécopie 01 64 15 31 07

Table des matières

Glossaire et sigles utiles

ENSG École Nationale des Sciences Géographiques
PPMD Mastère Spécialsé Photogrammétrie, Positionnement et Mesure de Déformations
IGN Institut National de l'Information Géographique et Forestière
LaSTIG Laboratoire en Sciences et Technologies de l'Information Géographique
MATIS Méthodes d'Analyses pour le Traitement d'Images et la Stéréorestitution
LiDAR Light Detection And Ranging
MNT Modèle Numérique de Terrain
MNS Modèle Numérique de Surface
LoD Level Of Details
SIG Système d'Information Géographique
SHP Shapefile
CSV Comma Separated Value
TIFF Tagged Image File Format

Introduction

J'introduis ...

Chapitre 1 Analyse du sujet

Une phase d'analyse du sujet a été préalablement réalisée. Ce premier chapitre a pour objectif de rappeler les éléments de contexte et la problématique du sujet. Il rappellera également les fonctionnalités initialement prévues dans le programme.

1.1 Définition du sujet

1.1.1 Contexte

Le projet s'inscrit dans le cadre des recherches du laboratoire MATIS de l'IGN pour la reconstruction 3D des bâtiments à partir de Modèles Numériques de Surface (MNS). Pour détecter et caractériser les erreurs de reconstruction, l'équipe de recherche a mis en place une méthode d'autoqualification. Ce processus passe par la réalisation d'une classification supervisée des entités reconstruites, afin de leur associer une classe d'erreur.

1.1.2 Problématique et analyse des besoins

L'objectif est d'évoluer vers une classification active. En effet, l'intervention d'un utilisateur permettrait de valider les résultats de la classification, afin d'affiner le modèle de prédiction entrainé.

Le projet de programmation présenté dans ce rapport est donc un outil graphique d'aide à la classification active. Grâce à une interface, l'utilisateur peut visualiser certains objets d'intérêt (orthophoto et bâtiments orthoprojetés) . Il peut ensuite valider le résultat de la classification, ou dans le cas contraire, renseigner la bonne erreur.

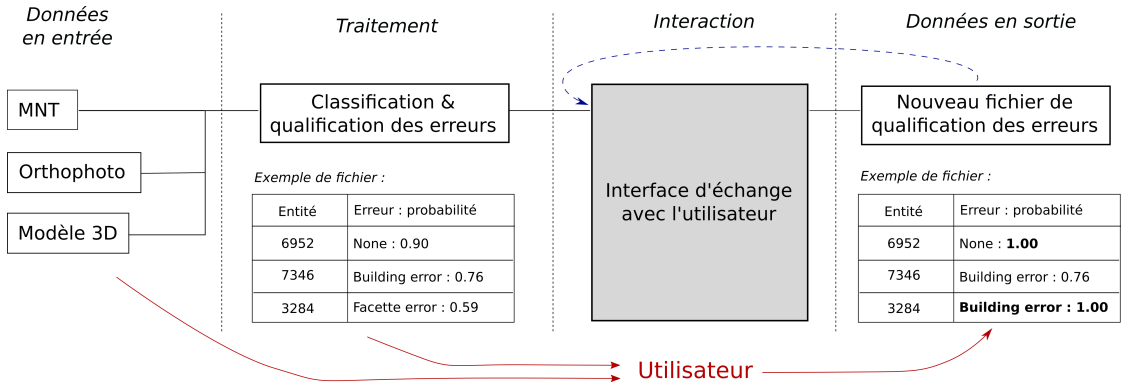

Figure 1.1 – Chaîne de traitement simplifiée du projet

Cette section a permis de retracer les principaux objectifs du projet. Le développement suivant vise à détailler les principales fonctionnalités envisagées lors de l'analyse.

1.2 Analyse fonctionnelle

1.2.1 Organisation des données

Le programme considère plusieurs données en entrée, sélectionnées par l'utilisateur via une première interface :

Données	Type de donnés	Correspondance dans le code
Classes d'erreurs possibles	Fichier .CSV	Variable de type dictionnaire
Résultats de l'auto-qualification	Fichier .CSV	Liste de tuples
Géométrie des entités	Dossier de .SHP	Attribut *geometry* de la classe Bâtiment
Orthophoto	Fichier .TIFF	Tuple (résolution + référence) + Matrice

FIGURE 1.2 – Données en entrée

De même, le formalisme des données en sortie est imposé :

Données	Type de donnés	Correspondance dans le code
Résultats de l'interaction	Fichier .CSV	Liste de tuples

FIGURE 1.3 – Données en sortie

1.2.2 Principales fonctionnalités

L'interface doit permettre à un utilisateur de visualiser et de contrôler les résultats d'une classification. Ainsi, les principales fonctionnalités à développer sont :

→ Une interface de chargement des données ;
→ Une fonction de sélection des entités à présenter à l'utilisateur ;
→ Une interface de visualisation graphique et textuelle des entités ;
→ Une fonction de contrôle des entités par l'utilisateur.

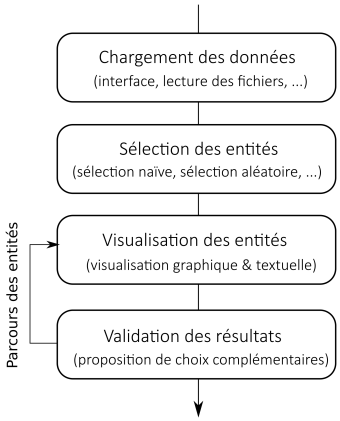

FIGURE 1.4 – Fonctionnalités principales

Après avoir rappelé les principales fonctionnalités du programme et les données nécessaires à son fonctionnement, on rappellera les choix techniques réalisés.

1.3 Choix techniques

L'interface graphique voulue pour ce projet utilise la bibliothèque Qt associée à un programme Python (PyQt5). C'est une solution indépendante et multiplateforme. Elle peut être exécutée sur tous les environnements, et elle ne dépend pas de l'évolution d'un autre framework, par exemple QGIS.

Chapitre 2 Implémentation de la solution

La phase d'analyse a permis de définir les objectifs du programme et d'organiser la phase de codage. Ce chapitre détaille les étapes de programmation effectivement réalisées, ainsi que les difficultés et les modifications ayant été apportée à la solution initiale.

2.1 Gestionnaire de version et environnement de développement

Avant de débuter la programmation, une étape d'organisation a été nécessaire avec le commanditaire. En effet, afin de communiquer facilement sur l'avancée du projet, un gestionnaire de version a été mis en place. Cela a permis d'enregistrer les évolutions du programme, de manière à pouvoir rappeler une version antérieure fonctionnelle à tout moment.

Le gestionnaire de version choisi est Git. Ce logiciel libre, distribué sous la licence GNU, est considéré comme l'état de l'art dans l'industrie. Une organisation *<activeML>* a été créée pour gérer facilement les projets sur GitHub. Ce système m'a permis d'interagir directement avec mon commanditaire en cas de problèmes ou de questions. Cependant, n'ayant jamais utilisé cet outil, sa prise en main a été compliquée.

Afin de faciliter la manipulation des commandes de Git, j'ai utilisé le plugin Git sur Atom. Développé par GitHub, cet éditeur de texte libre est très modulable.

2.2 Phases de programmation

Cette section vise à détailler les fonctionnalités générales à implémenter et les solutions apportées lors de la programmation.

2.2.1 Interface graphique

Comme défini lors de l'analyse, la méthode Modèle/Vue/Contrôleur (MVC) a été utilisée pour structurer le code. Ainsi, pour la partie *Vue*, trois interfaces graphiques ont été définies : une interface principale, une interface de chargement, et une interface pour corriger la classification. Ces trois interfaces ont été réalisées avec la bibliothèque multi-plateforme Qt.

La première phase de programmation a consisté à finaliser l'organisation de ces interfaces. Pour cela, l'environnement de développement Qt Creator a été choisi afin de faciliter la mise en place des objets. Le code issu de ce traitement a été transcrit en Python grâce à l'utilitaire Pyuic5. Il a ensuite fallu écrire un module permettant d'ouvrir ces fenêtres, et qui correspond à la partie *Contrôleur*.

Les principales difficultés rencontrées dans cette phase sont liées à la compréhension du code des interfaces. Dans le code principal, trois nouvelles classes ont été créées pour gérer les connexions entre les interfaces : *LoaderWindow*, *MainWindow*, *CorrectionWindow*. Ces classes héritent des trois classes initiales, créées par Pyuic5.

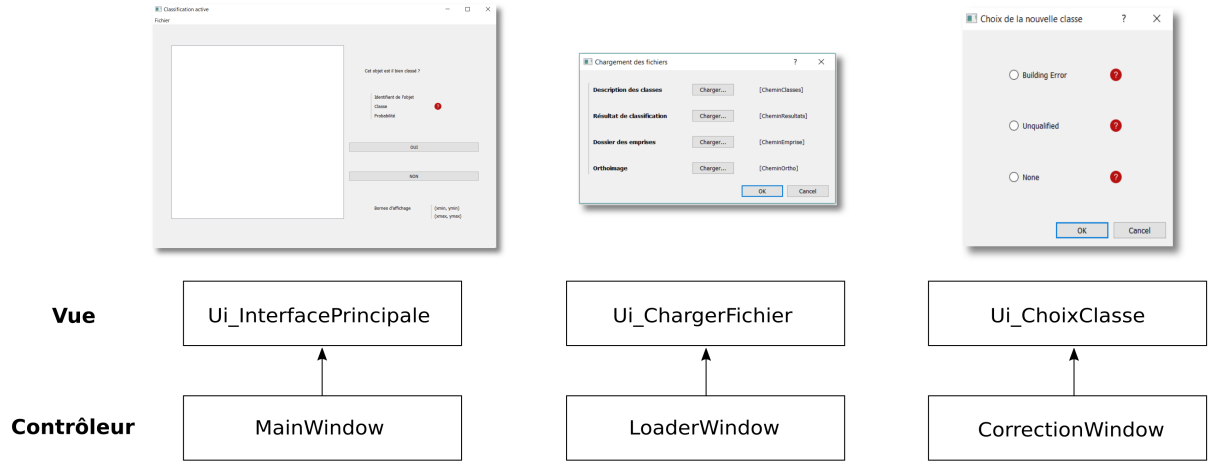

FIGURE 2.1 – Organisation des interfaces et héritages associés

2.2.2 Chargement des données

Pour permettre à l'utilisateur de charger les fichiers nécessaires au programme, une interface de chargement a été définie (*LoaderWindow*). Elle permet de sélectionner les 2 fichiers .CSV contenant les classes et les résultats de la qualification, le dossier contenant les emprises des entités et l'ortho-photo associée. Cette interface de chargement est lancée en cliquant sur *Fichiers* > *Charger...* , qui lance la fonction *show_loading_window*.

Les difficultés rencontrées lors de la programmation sont de deux ordres :

→ L'établissement de la connexion entre le bouton et l'ouverture de la fenêtre de chargement : ce problème a été résolu par l'utilisation de la classe *QFileDialog* de Qt. Ces connexions ont été implémentées dans la partie *Contrôleur* du programme.

→ L'enregistrement du chemin d'accès défini par l'utilisateur : après avoir réalisé des tests en créant une nouvelle classe *Path*, le choix s'est finalement porté sur l'intégration d'un attribut à la classe *LoaderWindow* pour chacun des chemins d'enregistrement.

Lorsque la phase de chargement est complétée, le programme principal peut s'exécuter. Comme prévu dans l'analyse, les caractéristiques des classes sont lues et enregistrées dans un dictionnaire, et les résultats de la qualification sont enregistrés dans une liste de tuples. L'orthoimage et les emprises ne sont lues que lors de l'affichage.

2.2.3 Sélection des données

Une fois l'étape de chargement des données effectuée, le programme en lui-même peut être lancé. Comme défini lors de l'analyse, l'étape suivante était de filtrer la liste des entités en entrée pour n'en présenter qu'un certain nombre à l'utilisateur. Cela s'apparente donc à la partie *Modèle* du programme.

L'analyse préconisait la création d'une classe abstraite *Strategy* permettant l'application du filtre. Les types de stratégies devaient être définies dans des classes qui héritent de ces propriétés. Lors de la programmation, il est apparu que les classes abstraites en Python ne font pas parties du cœur même de Python, et sont accessibles grâce à la bibliothèque *abc* (Abstract Base Classes).

Pour simplifier la programmation, la classe mère *Strategy* n'a pas été implémentée comme abstraite. Les classes filles héritent simplement des propriétés de la classe mère et viennent les compléter.

Dans un premier temps, seules les stratégies *Naïve* (sélection de toutes les entités) et *Random* (sélection d'un nombre aléatoire d'entités) ont été implémentées.

Une amélioration a été apportée au programme initialement imaginé. Elle permet à l'utilisateur de sélectionner la méthode de filtrage qu'il souhaite utiliser dans l'interface de chargement des fichiers. Pour ce faire, une variable globale *STRATEGIES* regroupe, sous forme de dictionnaire, l'ensemble des classes filles de *Strategy*. Cette variable est créée en utilisant la réflexion, c'est à dire en utilisant la capacité de Python à examiner ses propres structures internes.

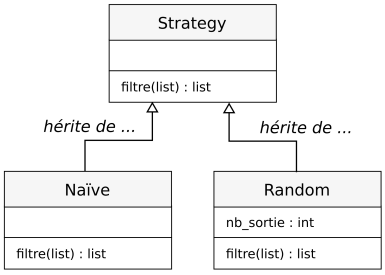

FIGURE 2.2 – Stratégies de sélection

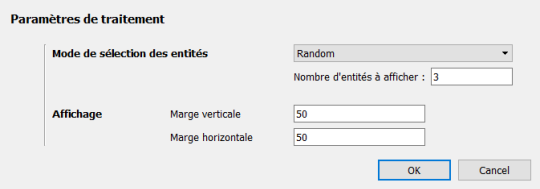

FIGURE 2.3 – Sélection de la stratégie dans l'interface de chargement

2.2.4 Visualisation des entités

A ce point de la programmation, on obtient une liste de tuples correspondant aux objets à montrer à l'utilisateur. Lors de l'analyse, plusieurs étapes ont été définies pour afficher les différentes entités :

→ Création d'une liste d'objets *Bâtiment* ;

→ Calcul des limites d'affichage ;

→ Préparation de l'affichage de l'orthoimage et des emprises ;

→ Affichage des caractéristiques de l'entité.

On détaillera chacune de ces étapes et les problématiques qui lui sont liées.

Création d'une liste d'objets *Bâtiment*

Une classe *Building* a été créée pour enregistrer l'identifiant, la géométrie, la classe et la probabilité associée à chaque entité à afficher. Pour renseigner les attributs d'identifiant, de classe et de probabilité, on utilise simplement les valeurs du fichier des résultats de la classification.

Pour le renseignement de la géométrie des objets, l'implémentation a été plus compliquée. Les données de géométrie étaient initialement au format .GML. Cependant, la lecture de ces fichiers n'était pas gérée par le package *gdal*. Les emprises au format .SHP ont donc été privilégiées. Pour lire la géométrie contenue dans ces fichiers, la fonction *read_building* a été implémentée. Elle se base sur les fonctions du package *pyshp*.

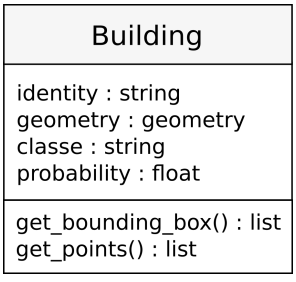

FIGURE 2.4 – Classe Building

Calcul des limites d'affichage

La détermination de la fenêtre d'affichage se base sur les coordonnées des extrémités des emprises. Pour les calculer, deux méthodes ont été créées dans la classe *Building* : *get_points* qui récupère les points de la géométrie et *get_bounding_box* qui sélectionne les maxima et minima. Lors de l'affichage, les marges entrées par l'utilisateur sont prises en compte. Un problème est apparu lorsque la marge dépasse l'image : une translation est donc réalisée.

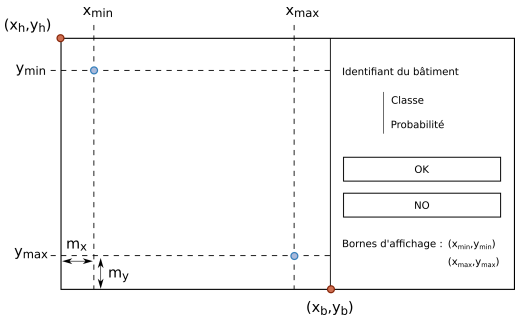

FIGURE 2.5 – Limites d'affichage

Affichage de l'orthoimage

La lecture de l'orthoimage a nécessité plus d'étapes que prévues lors de l'analyse. Pour mieux organiser le code, une classe *Background* a été créée. Elle prend comme attribut les coordonnées du point de référence de l'orthophoto, sa taille de pixel et la matrice d'image. Le package *gdal* a été utilisé pour lire les données issues du fichier .TIFF.

Pour charger l'orthoimage et créer l'objet *Background*, la fonction *read_background*, a été implémentée. Cette fonction est issue du programme d'auto-qualification réalisé par Oussama ENNAFII, commanditaire du projet. Pour rogner l'orthoimage aux dimensions de l'entité, une méthode *crop* a été ajoutée à la classe *Background*. Elle retourne l'image couleur rognée.

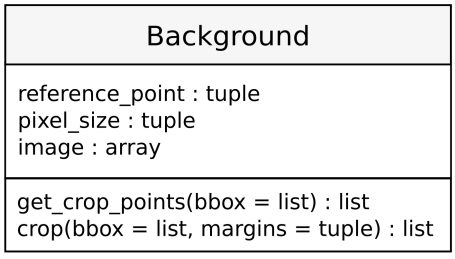

FIGURE 2.6 – Classe Background

La phase d'affichage a nécessité l'utilisation des méthodes de l'objet *QGraphicsView* de Qt. Des options particulières ont permis d'adapter la scène à la fenêtre graphique.

Affichage des emprises

L'affichage des géométries des entités a nécessité l'utilisation des méthodes des classes *QPolygonF* et *QPointF* de Qt. La principale difficulté a été de superposer l'entité à l'orthoimage en appliquant correctement les valeurs des marges.

Affichage des caractéristiques de l'entité

L'affichage des caractéristiques de l'entité est basée sur la lecture des attributs l'objet *Building*. Des fonctionnalités supplémentaires ont été développées, comme l'affichage des coordonnées des bornes, ou l'inclusion d'infobulles expliquant le type de classe.

2.2.5 Interaction avec l'utilisateur

Cette étape de la programmation correspond à la partie *Contrôleur* du code. Grâce à l'affichage des entités et de leurs caractéristiques, l'utilisateur peut valider le résultat ou sélectionner une autre classe.

Initialement, une seule fonction était prévue pour cette phase d'interaction. Mais suite à des difficultés pour parcourir la liste des entités à présenter, un nouveau formalisme a été imaginé. Trois fonctions ont donc été créées :

→ La fonction *next()* permet de parcourir la liste des entités à présenter et d'afficher son emprise et ses caractéristiques ;

→ La fonction *validate()* permet d'enregistrer l'entité présentée comme valide ;

→ La fonction *correct()* lance l'interface de sélection d'une nouvelle classe et enregistre le choix de l'utilisateur.

Lorsque la liste des entités à présenter est vide, la phase d'interaction avec l'utilisateur terminée. Une dernière fonction *save()* est lancée. Elle permet de demander à l'utilisateur le chemin d'enregistrement des résultats. Si celui-ci n'est pas correctement défini, une fenêtre d'erreur s'affiche. Chaque entité visualisée est ensuite enregistrée dans un fichier .CSV, dont le formalisme est identique aux données en entrée.

Cette section a présenté les différentes étapes de programmation et les modifications apportées. En effet, la phase d'analyse avait permis de définir le cadre général du programme, mais son implémentation a mis à jour des difficultés non envisagées initialement.

2.3 Planning opérationnel

Bien que plusieurs créneaux horaires aient été dédiés à la phase de programmation, le planning envisagé lors de la phase d'analyse n'a pas été complètement suivi.

Séance	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	+
Finalisation et connexion des interfaces																
Chargement des données																
Définition des fonctions de filtrage																
Lecture et affichage des orthoimages																
Lecture et affichage des emprises .SHP																
Interaction avec utilisateur																
Amélioration et mise en forme du code																
Rédaction des rapports																

FIGURE 2.7 – Planning suivi pour la phase de programmation

Chapitre 3 Limites et évolutions de la solution

La phase de programmation a permis de développer une première solution au sujet posé. Cependant, certaines fonctionnalités manquent encore. Ce chapitre détaille donc les améliorations pouvant être apportées au programme.

3.1 Limites de la solution proposée

Bien que la solution proposée soit fonctionnelle et réponde à la problématique du projet, certaines fonctionnalités manquent encore :

→ Fonction de zoom : pour permettre à l'utilisateur d'agrandir ou de réduire l'entité présentée, une fonction de zoom peut être implémentée. Cette fonctionnalité pourra se baser sur les méthodes de la classe *QGraphicsView*, et sur une fonction enregistrant les rotations de la molette de la souris.

→ Gestion des marges trop grandes : dans la fenêtre de chargement, l'utilisateur peut définir des marges pour visualiser l'environnement des bâtiments. Pour l'instant, aucune fonctionnalité ne gère les cas de marges dépassant les bordures de l'orthoimage.

→ Affichage des MNS : grâce aux méthodes de la classe *QPixMap*, l'interface peut afficher une orthoimage couleur, constituée de trois bandes. Une nouvelle fonctionnalité pourrait gérer le cas de l'afficher des MNS constitués d'une seule bande.

De plus, l'implémentation de cette solution a été réalisée en s'appuyant sur un jeu de données types. Leur formalisme étant imposé à l'utilisateur, le programme devrait fonctionner avec d'autres données. Cependant, des tests complémentaires pourraient être réalisés pour vérifier la compatibilité de l'interface (utilisation d'une autre orthophoto, utilisation de plus d'emprises, ...).

3.2 Évolution de la solution

- Autres méthodes de filtrage
- Autre type de classification

Conclusion

Je conclus.

Table des figures