



ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES



Université Paris-Est

Ecole Nationale des
Sciences Géographiques

-PROJET INFORMATIQUE -
Rapport de programmation

Mastère spécialisé ® Photogrammétrie, Positionnement, Mesure de Déformations

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



ÉCOLE NATIONALE
DES SCIENCES
GÉOGRAPHIQUES

CHUPIN Clémence

Novembre-Décembre 2017

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

ÉCOLE 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 | 4 |
| Introduction | 5 |
| 1 Analyse du sujet | 6 |
| 1.1 Définition du sujet | 6 |
| 1.2 Analyse fonctionnelle | 7 |
| 1.3 Choix techniques | 7 |
| 2 Implémentation de la solution | 8 |
| 2.1 Gestionnaire de version et environnement de développement | 8 |
| 2.2 Phases de programmation | 8 |
| 2.3 Planning opérationnel | 11 |
| 3 Limites et évolutions de la solution | 12 |
| 3.1 Limites de la solution proposée | 12 |
| 3.2 Fonctionnalités à développer | 12 |
| Conclusion | 13 |

Glossaire et sigles utiles

ENSG École Nationale des Sciences Géographiques

PPMD Mastère Spécialisé 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 ...

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'auto-qualification. 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 entraîné.

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 de renseigner la bonne erreur.

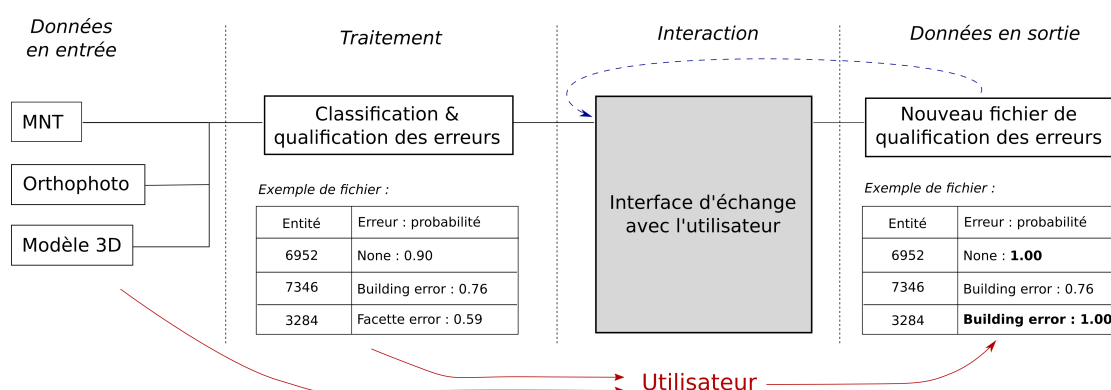


FIGURE 1.1 – Chaîne de traitement simplifiée du projet

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 <i>geometry</i> 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.

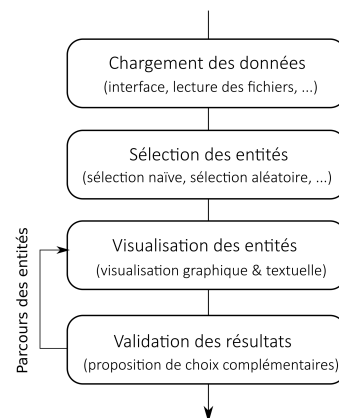


FIGURE 1.4 – Fonctionnalités principales

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 script Python (PyQt5). Cela lui permet de fonctionner sur tous les environnements informatiques, et elle peut être modulée sans prendre en compte le formalisme de QGIS.

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 apportées à la solution initiale.

2.1 Gestionnaire de version et environnement de développement

Avant de débiter 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 de gestion de versions, distribué sous licence libre, est très largement utilisé et reconnu, ce qui justifie son utilisation. Afin d'échanger facilement avec le commanditaire, un projet "activeML" a été créé sur le site 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. Un problème de fichiers trop volumineux lors d'un *commit* a notamment nécessité de prendre en main la commande *git rebase*. Au final, plusieurs heures auront été nécessaires à la résolution du problème. Ce temps perdu sur la programmation m'aura cependant permis de gagner en efficacité par la suite.

Afin de faciliter la manipulation des commandes de Git, l'éditeur de texte libre Atom a été utilisé. Développé par GitHub, cet éditeur offre la possibilité d'installer de nombreuses extensions, et il facilite l'accès aux commandes de Git (*commit*, *pull*, *push*, ...).

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

Lors de l'analyse du projet, 3 interfaces graphiques ont été définies : une interface principale, une interface de chargement, et une interface pour corriger la classification. Ces 3 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 le script permettant d'ouvrir ces fenêtres.

Les principales difficultés rencontrées dans cette phase sont liées à la compréhension du code des interfaces. Dans le code principal, 3 nouvelles classes ont été créées pour gérer les connexions entre les interfaces : *LoaderWindow*, *MainWindow*, *CorrectionWindow*. Ces classes héritent des 3 classes initiales, créées par Pyuic5.

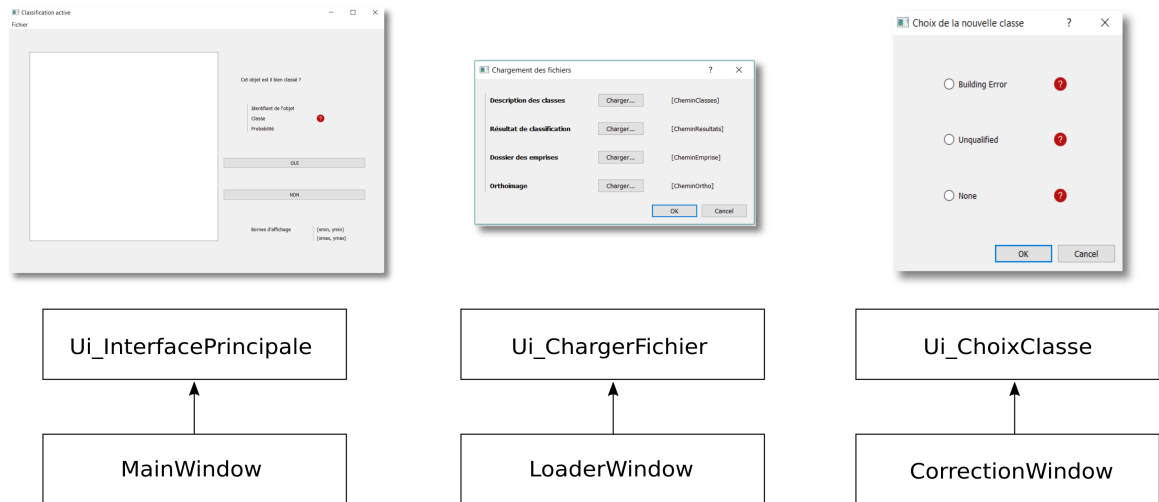


FIGURE 2.1 – Organisation des interfaces et héritages associés

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.
- 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.

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 surcharger la fonction de filtre.

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.

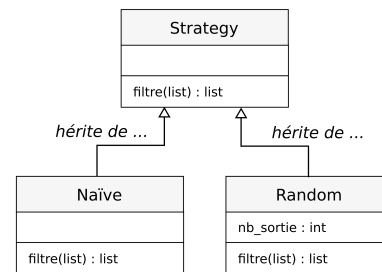


FIGURE 2.2 – Stratégies de sélection

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*.

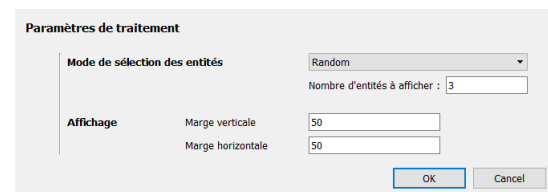


FIGURE 2.3 – Sélection de la stratégie dans l'interface de chargement

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
- 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 était compliquée et nécessitait l'installation de bibliothèques non intégrées à Python. 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 de la bibliothèque *shapefile*.

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, il faudra prendre en compte les marges entrées par l'utilisateur.

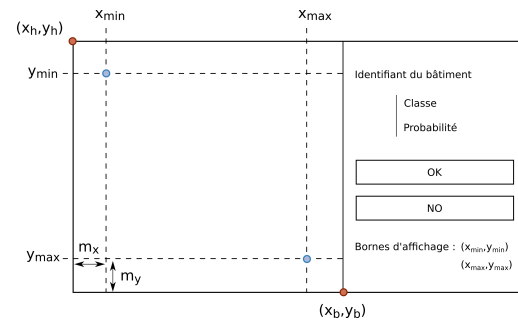


FIGURE 2.4 – Limites d'affichage

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. La bibliothèque *gdal* a été utilisée 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.

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

- lecture de toutes les entités building de la liste initiale (forme de récurrence)
- selon le choix de l'utilisateur, lancement de la fonction correct ou valide
- sauvegarde des résultats avec la fonction save
- gestion des conflits : clic sur oui/non par l'utilisateur avant chargement, chemin d'enregistrement non défini

2.3 Planning opérationnel

LIMITES ET ÉVOLUTIONS DE LA SOLUTION

CHAPITRE 3

Le présent chapitre a pour objectif de ...

3.1 Limites de la solution proposée

- Certaines variables sont codées "en dur" (affichage des fonction de sélection, ...)
- Tests avec plusieurs jeux de données
- ...

3.2 Fonctionnalités à développer

- Autres méthodes de filtrage
- Classification active
- ...

Conclusion

Je conclus.

Table des figures

| | | |
|-----|--|----|
| 1.1 | Chaîne de traitement simplifiée du projet | 6 |
| 1.2 | Données en entrée | 7 |
| 1.3 | Données en sortie | 7 |
| 1.4 | Fonctionnalités principales | 7 |
| 2.1 | Organisation des interfaces et héritages associés | 9 |
| 2.2 | Stratégies de sélection | 10 |
| 2.3 | Sélection de la stratégie dans l'interface de chargement | 10 |
| 2.4 | Limites d'affichage | 11 |