Mémoire

Auteur : Troussart, Evelyne
Promoteur(s) : Billen, Roland; Poux, Florent
Faculté : Faculté des Sciences
Diplôme : Master en sciences géographiques, orientation géomatique, à finalité spécialisée en géomètre-expert
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/12535

Faculté des sciences
Département de géographie

Méthodes de géocodification par scanner laser 3D terrestre

Mémoire présenté par : **Evelyne TROUSSART**

pour l'obtention du titre de

Master en sciences géographiques,

orientation géomatique, à finalité spécialisée en géomètre-expert

Année académique :	**2020-2021**
Date de défense :	**Septembre 2021**
Président de jury :	Pr. **René WARNANT**
Promoteurs :	Pr. **Roland BILLEN** Pr. **Florent POUX**
Jury de lecture :	Pr. **René WARNANT** Dr. **Romain NEUVILLE**

Mes premiers remerciements vont à mes promoteurs, Messieurs Roland Billen et Florent Poux, pour leurs conseils et leur accompagnement tout au long de ce travail.

Je tiens également à remercier mes lecteurs, Messieurs René Warnant et Romain Neuville, pour l'intérêt porté à mon mémoire.

J'adresse ensuite un grand merci à mes parents pour leurs relectures et pour leur soutien durant ces cinq années d'étude.

Enfin, je tiens à remercier mes amis et mes proches pour leurs encouragements et plus particulièrement Thomas Dethinne pour son aide, sa patience et son soutien au cours de ces dernières années.

Résumé

L'utilisation des nuages de points s'est accrue au cours des dernières années. Afin d'exploiter correctement ces données, la segmentation et la classification semblent essentielles et font dès lors l'objet de nombreuses recherches. La plupart de ces recherches se concentrent uniquement sur le post-traitement des données. Il semble toutefois intéressant de considérer l'intégration d'informations sémantiques lors de l'acquisition pour faciliter ces traitements. C'est le principe de la géocodification, principalement utilisée par les géomètres lors de leurs levés.

Ce travail envisage donc l'utilisation d'une géocodification lors de l'acquisition d'un nuage de points avec l'objectif d'améliorer le processus de segmentation/classification d'un environnement bâti grâce aux informations récoltées.

Deux méthodologies différentes ont été développées à cet effet. La première se rapproche d'une géocodification traditionnelle puisqu'elle consiste à combiner les informations spatiales et sémantiques issues d'un levé topographique avec un nuage de points afin de procéder à sa classification. La seconde méthode, plus innovante, se base sur le positionnement de cibles sur les éléments d'intérêts afin de guider le processus de segmentation/classification. Les différentes étapes nécessaires à la réalisation de ces deux méthodes sont détaillées dans ce travail, depuis l'établissement des géocodifications jusqu'à l'obtention des nuages de points classifiés, en passant par l'acquisition et les différents prétraitements à appliquer.

Les résultats obtenus pour les deux méthodes sont ensuite présentés, validés et comparés entre eux. Il en ressort que les deux méthodes proposées sont plus rapides mais un peu moins précises qu'une segmentation manuelle avec un F1-score moyen compris entre 0,82 et 0,93 pour le nuage étudié. La méthode avec levé topographique présente des résultats de meilleure qualité que la méthode des cibles mais elle nécessite un temps d'acquisition plus long. Ce travail se clôture sur la mise en évidence des limites rencontrées pour les deux méthodes. Des perspectives d'amélioration et de développement sont également présentées.

Abstract

The use of point clouds has increased in recent years. In order to exploit these data correctly, segmentation and classification appear essential and are therefore the subject of numerous researches. Most of these researches only focus on post-processing data. However, it seems interesting to consider the integration of semantic information during the data acquisition to facilitate the treatments. This is the principle of feature codes, mainly used by land surveyors during their surveys.

Therefore, this work considers the use of feature codes during a point cloud acquisition with the aim of improving the segmentation/classification process of a built environment through the collected information.

Two different methodologies have been developed for this purpose. The first one is similar to traditional feature codes since it consists in combining spatial and semantic information from a topographical survey with a point cloud in order to proceed with its classification. The second method, more innovative, is based on the positioning of targets on the elements of interest to guide the segmentation/classification process. The different steps necessary to achieve these two methods are detailed in this work, from the establishment of the feature codes list to the point clouds classification, passing by the acquisition and the pre-processing.

The obtained results for both methods are then presented, validated and compared. It shows that the two proposed methods are faster but slightly less accurate than a manual segmentation with an average F1-score between 0.82 and 0.93 for the point cloud studied. The topographic survey method presents better results than the target method but it requires a longer acquisition time. This work concludes by highlighting the limitations encountered for the two methods. Improvement and development perspectives are also presented.

Table des matières

Liste des figures

Liste des tableaux

1. INTRODUCTION

Le développement des scanners laser a conduit à une utilisation accrue des nuages de points dans de nombreux domaines tels que la robotique, l'ingénierie, la construction, l'architecture ou encore l'archéologie (Biosca & Lerma, 2008 ; Poux *et al.,* 2016). Avec cette technique, quelques secondes suffisent pour acquérir des millions de points positionnés dans l'espace avec précision. Cependant, une telle quantité de données non-structurées et sans informations sémantiques nécessitent toute une série de traitements afin d'être exploitées efficacement. Parmi ces traitements, la segmentation et la classification du nuage apparaissent essentielles (Poux *et al.*, 2014). La segmentation consiste à regrouper les points aux propriétés similaires alors que la classification est définie comme l'attribution d'une classe spécifique à ces points (Grilli *et al.*, 2017). La segmentation manuelle étant particulièrement chronophage et fastidieuse, de nombreux travaux se sont attachés à l'automatisation de ce processus (Nguyen & Le, 2013).

Néanmoins, pour réaliser cette segmentation, la plupart des auteurs se concentrent uniquement sur le post-traitement du nuage de points, sans informations à priori. Il semble pourtant intéressant d'envisager l'apport que pourrait avoir une acquisition réfléchie sur ce processus de segmentation. En effet, lors d'un levé topographique, il est courant d'utiliser une géocodification afin d'associer directement une information sémantique à une information spatiale et ainsi faciliter les post-traitements tels que le dessin du plan. Ce principe pourrait dès lors être appliqué lors de l'acquisition d'un nuage de points pour faciliter ensuite le processus de segmentation et de classification.

L'objectif de ce travail est donc de créer et tester une géocodification à utiliser lors de l'acquisition de données au laserscan. A partir de ces données, un algorithme de segmentation sera développé et les résultats obtenus seront comparés avec ceux d'une segmentation manuelle. Ainsi, la question à la base de ce mémoire est la suivante :

> « Est-il possible d'utiliser une géocodification lors de l'acquisition d'un nuage de points par laserscan pour améliorer le processus de segmentation et classification ? »

Afin de répondre à cette question, deux méthodes différentes sont présentées. La première se rapproche du travail traditionnel d'un géomètre puisqu'elle cherche à combiner les données spatiales et sémantiques issues d'un levé topographique avec un nuage de points de manière à

procéder à sa segmentation. La seconde méthode, plus innovante, consiste à positionner des cibles de manière spécifique lors de l'acquisition du nuage de points et d'effectuer la segmentation à partir de ces cibles.

Ce travail se divise comme suit. Le chapitre 2 est consacré à la définition et à l'explication des différents concepts importants liés à ce sujet. Il présente également un aperçu de la littérature relative aux méthodes de segmentation d'un nuage de points, avec et sans intégration de données topographiques. Sur base de cet état de l'art, les hypothèses de recherche sont formulées dans le 3e chapitre. Le chapitre 4 détaille quant à lui la méthodologie suivie pour le développement des deux méthodes ainsi que pour leur validation. L'environnement de travail est ensuite repris dans le chapitre 5 alors que les différents résultats sont présentés dans le 6e. Enfin, le chapitre 7 présente une discussion mettant en évidence les avantages et les limites des méthodes proposées tandis que le chapitre 8 conclut ce travail.

2. ETAT DE L'ART

2.1. Définitions et concepts de base

2.1.1. Levé topographique et géocodification

Le levé topographique est l'opération qui consiste à recueillir et enregistrer des informations géométriques et thématiques sur un terrain donné à l'aide d'instruments adaptés. Il permet de définir la localisation des éléments de la zone étudiée dans un même système de référence. Il est à la base de l'établissement de plans ou de cartes, représentations de la réalité à l'échelle souhaitée par son auteur. A l'aide de mesures d'angles et de distances effectuées à la station totale (ou tachéomètre) par exemple, les coordonnées des points d'intérêt peuvent être déterminées.

Outre l'enregistrement de données géométriques, il est possible d'associer une information thématique, plus particulièrement un code, rendant compte de la nature du point levé. Il s'agit de la géocodification. La géocodification consiste à définir au préalable une série de codes qui seront attribués aux points mesurés lors du levé (Billen, 2018 ; Jonlet & Poux, 2017). Il faut dès lors tenir compte des éléments pouvant être rencontrés sur le terrain pour établir la table de géocodification. Elle est essentiellement utilisée pour gagner du temps et de la clarté pendant la prise de mesures puisqu'il n'est pas nécessaire de tout consigner dans un carnet de terrain. Elle permet également de faciliter l'élaboration du plan de la zone étudiée. En effet, l'utilisation d'un moteur de géocodification, capable d'interpréter une série de commandes associées à chaque code (symbole, orientation, etc.), va permettre la génération d'un plan plus ou moins abouti pour lequel il suffira de corriger les quelques erreurs présentes et d'ajouter certaines données complémentaires.

En ce qui concerne la géocodification en tant que telle, quelques notions sont importantes à savoir. Tout d'abord, plusieurs codes peuvent être attribués à un même point grâce à l'utilisation d'un séparateur de code. C'est par exemple le cas lorsque l'on se trouve à une intersection, l'extrémité d'une maison pouvant notamment coïncider avec le début d'une haie. Ensuite, il est important de distinguer les différents types d'objets : éléments ponctuels, linéaires ou zonaux. Les objets ponctuels (lampadaire, taque d'égout, etc.) sont les plus simples puisqu'ils sont levés en un point, parfois en deux ou trois points si on veut l'emprise complète de l'objet, avec un code « unique » et l'ordre dans lequel ils sont mesurés n'a pas

d'importance. Les objets linéaires et zonaux peuvent être considérés ensemble, un objet zonal pouvant être vu comme une polyligne fermée. Le code associé à ces éléments s'appelle une liaison, elle-même décomposée en séquence et modificateur. La séquence correspond à la partie principale du code définissant la nature de l'objet. Les modificateurs sont indépendants de la nature, ils spécifient la partie géométrique du segment au niveau du point levé, c'est-à-dire s'il s'agit du début d'une séquence droite ou courbée ou encore d'un point collecté au milieu de la séquence. Pour illustrer ces différents termes, considérons le levé d'une route dont la séquence serait 100. Le point de début d'une route droite se verrait par exemple attribuer le code 100.2, un point levé plus loin sur cette route serait 100.4 ; si elle commence à tourner, le code 100.3 serait appliqué aux points mesurés tout au long du virage. Un modificateur particulier (généralement .0) permet de refermer l'entité sur son premier point et ainsi constituer une entité zonale. Pour ces deux types d'objets, il est primordial de lever les points dans l'ordre dans lequel ils seront reliés (Jonlet & Poux, 2017).

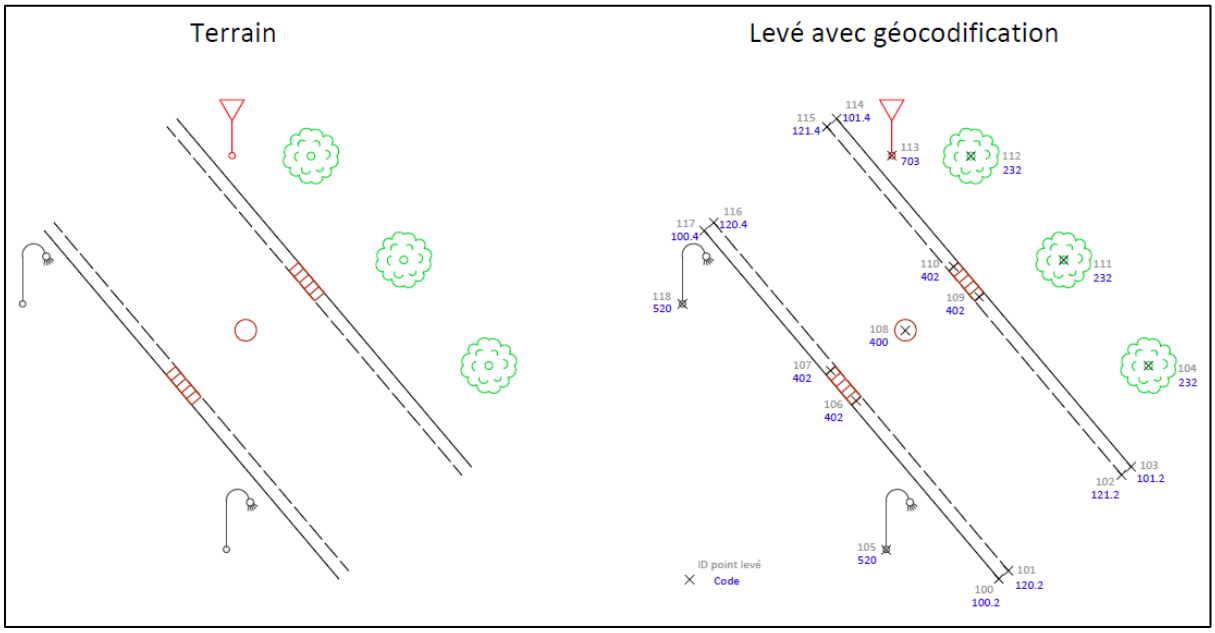


Figure 1 : Levé topographique et géocodification

2.1.2. Nuage de points : acquisition et traitements

Un nuage de points est un ensemble de points tridimensionnels représentant la surface des éléments relevés. Il est caractérisé par une densité, nombre de points acquis par unité de surface, dépendant de la résolution fixée par l'utilisateur et de la distance à l'appareil de mesure (Landes & Grussenmeyer, 2011). En plus des coordonnées x, y et z, une intensité et/ou une information de couleur RVB (rouge, vert, bleu) peut être associée à chaque point du

nuage. Poux *et al.* (2014) caractérisent les nuages de points comme des données complexes volumineuses, souvent bruitées et dans lesquelles il subsiste des zones d'ombres, c'est-à-dire des zones qui n'ont pu être mesurées lors de l'acquisition (occlusions).

2.1.2.1. Acquisition et principe de mesure

D'après Xie *et al.* (2020), il existe quatre méthodes principales pour acquérir des nuages de points : la photogrammétrie, les caméras de profondeur, les systèmes SAR (*Synthetic Aperture Radar*) et la technique LIDAR (*Light Detection And Ranging*). Cette dernière se base sur le temps de parcours d'un rayon laser pour déterminer la distance entre l'objet étudié et le capteur. Les auteurs subdivisent cette technique en plusieurs catégories selon le type d'appareil utilisé. En effet, il existe des scanners laser aériens (ALS), terrestres (TLS), mobiles (MLS) ou encore par drone (ULS). Chacun de ses systèmes a évidemment ses avantages et ses inconvénients. La résolution des scanners laser aériens est nettement plus basse que celle des MLS ou TLS, ils ne permettent pas la modélisation des détails et petits objets. Néanmoins, ils permettent de lever rapidement une grande étendue. Une autre caractéristique des ALS (et des ULS) concerne la géométrie de la prise de mesures, les données collectées proviennent principalement de la surface supérieure des objets alors que les côtés sont peu représentés. A l'inverse, les scanners laser terrestres et mobiles se concentrent plutôt sur les données de façades et sont notamment limités par la hauteur des bâtiments. La densité et la précision des données récoltées par un scanner laser terrestre sont élevées mais sa position statique limite l'étendue de la zone mesurée (Cabo *et al.*, 2015). Plusieurs stations sont nécessaires pour mesurer les éléments dans leur intégralité et éviter au maximum les zones d'ombres. Le système MLS, opérant depuis un véhicule ou une personne en mouvement, est un peu moins précis mais il permet de scanner des zones plus larges tout en limitant le temps d'acquisition (Yang *et al.*, 2015 ; Che *et al.*, 2019 ; Xie *et al.*, 2020). Il faut donc choisir la ou les techniques adaptées en fonction du but poursuivi.

Le balayage laser terrestre ou système TLS utilise la lumière laser pour déterminer les coordonnées tridimensionnelles des points autour d'une position terrestre, celle de l'appareil de mesure. Il permet de collecter une quantité de points très élevée en un temps limité. Il existe différents types de scanners classés en fonction du principe de mesure de la distance entre l'objet et l'instrument de mesure. Landes & Grussenmeyer (2011) définissent trois catégories: scanners laser à impulsions, scanners à différence de phase et scanners à triangulation.

Le principe des scanners à impulsions est de déterminer une distance sur base de la mesure du temps de parcours d'un pulse laser, temps entre l'émission et la réception de ce pulse après réflexion sur l'objet. La relation suivante permet de déterminer la distance objet-scanner (D) à partir de la vitesse de la lumière (du faisceau laser) (v) et de ce temps de parcours (t) :

$$2D = v * t \quad (1)$$

Dès lors, cette technique nécessite l'emploi d'horloges de haute précision, une petite erreur sur le temps de parcours ayant des conséquences importantes sur la distance calculée. Ce type de scanner est adapté pour de longues portées mais sa vitesse d'acquisition est limitée par la nécessité de réceptionner le pulse émis avant de pouvoir effectuer la mesure suivante.

Les scanners à différence de phase déterminent la distance objet-scanner sur base de la différence de phase (également appelée déphasage) mesurée entre le signal émis et le signal reçu. Pour ce faire, un signal continu modulé de manière sinusoïdale est émis vers l'objet. L'onde reçue après contact avec l'objet est ensuite comparée à l'onde de départ. La distance totale parcourue (2D) vaut alors un nombre entier de longueurs d'onde (N) auquel il faut ajouter un excédent dû au déphasage ($\Delta\varphi$, différence de phase entre les deux ondes) :

$$2D = N\lambda + \frac{\Delta\varphi}{2\pi}\lambda \quad (2)$$

L'indétermination sur N, appelée ambiguïté initiale, doit être levée afin de résoudre l'équation. Plusieurs méthodes existent pour lever cette indétermination, par exemple en émettant à plusieurs fréquences proches ou en superposant des ondes de différentes longueurs d'onde. Cette technique est plus rapide que la précédente mais sa portée est limitée par la nécessité d'avoir une intensité forte du signal en continu.

La dernière catégorie reprend les scanners à triangulation, utilisés généralement pour les objets de petite taille. Le dispositif récepteur est décalé par rapport à l'émetteur afin de former un triangle avec le point mesuré. Connaissant la distance entre l'émetteur et le récepteur (d_{AB}), le scanner mesure les angles α et β, respectivement formés par le faisceau émis et reçu avec la verticale afin de calculer la distance objet-scanner (D) :

$$D = d_{AB} * \frac{\sin\beta}{\sin(\alpha+\beta)} \quad (3)$$

Ces scanners sont limités à de courtes distances car l'imprécision sur la distance augmente avec le carré de celle-ci.

Outre la distance, les angles horizontaux et verticaux sont également mesurés afin de positionner les points levés dans un système tridimensionnel.

2.1.2.2. Géoréférencement et consolidation

Il est généralement nécessaire d'effectuer plusieurs scans pour mesurer l'objet d'intérêt dans son intégralité. En effet, un seul scan signifie un seul point de vue ce qui peut être insuffisant pour la dimension de l'objet. Il peut également y avoir des obstacles (arbres, véhicules, etc.) limitant le champ de vision. Chaque nuage de points est initialement référencé dans un système relatif à la position du scanner. L'assemblage des différents nuages de points dans un même système de coordonnées est alors indispensable pour obtenir l'intégralité de l'objet ou de la zone d'intérêt. Il existe deux types de géoréférencement : le géoréférencement direct et le géoréférencement indirect (Van Genechten, 2008 ; Landes *et al.*, 2011).

Le géoréférencement direct consiste à déterminer directement les coordonnées des points du nuage. Pour que cette solution soit possible, il faut que le scanner utilisé puisse être centré sur un point de référence et s'orienter sur base d'un autre point connu. Le scanner est alors assimilé à une station totale et une polygonale liant les différents points de vue (stations de scan) peut être établie. Le nuage est géoréférencé dans le système de coordonnées souhaité lors de l'acquisition et non en post-traitement. L'avantage principal de cette méthode est qu'elle ne nécessite pas un recouvrement important entre les différents nuages, contrairement à la méthode indirecte, puisque chaque nuage est directement positionné dans un système unique. Un des problèmes principaux du traitement de nuages de points étant le volume de données récoltées, celui-ci peut être en partie réduit par ce recouvrement limité. Cependant, l'imprécision sur les points de référence impacte directement la qualité de l'assemblage des différents nuages de points. Il faut donc veiller au bon positionnement du scanner vis-à-vis de ces points et tenir compte de leur fiabilité/précision.

Le géoréférencement indirect nécessite une opération préalable, appelée consolidation, lors de laquelle les différents nuages de points sont assemblés ensemble. Cette consolidation, illustrée à la figure 2, s'effectue sur base de points homologues entre les nuages, il faut donc un recouvrement suffisant entre ceux-ci. Ces points homologues peuvent être artificiels (cibles) ou naturels, c'est-à-dire des points présents dans la scène facilement identifiables. Il faut minimum trois points pour déterminer la transformation à appliquer à un des nuages pour le recaler sur l'autre. L'idéal est tout de même d'avoir plus de trois points pour assurer une redondance dans les données et minimiser les erreurs en ajustant les paramètres de la

transformation par moindres carrés. La qualité de la consolidation effectuée à l'aide de cibles, plates ou sphériques, dépend de leur distribution spatiale et du nombre de cibles communes entre les scans. Le positionnement de ces cibles est donc une étape essentielle pour obtenir des résultats satisfaisants. Après l'acquisition, le logiciel de traitement identifie les centres des cibles dans les différents nuages et procède à leur assemblage sur base des correspondances trouvées. S'il n'est pas possible de placer des cibles (zones difficilement accessibles par exemple), la consolidation peut être effectuée en sélectionnant manuellement des points identifiables avec précision. Néanmoins, le résultat sera moins précis puisque leur identification est relativement subjective et moins nette.

Il existe également des méthodes automatiques effectuant la consolidation sur base d'une recherche de points homologues (méthode ICP – *Iterative Closest Point*) ou une recherche d'entités homologues. La première part du principe que, suite aux pas horizontaux et verticaux des nuages à assembler, il est peu probable d'avoir mesuré exactement le même point dans les deux nuages. Pour contourner ce problème, la méthode ICP va chercher le point correspondant le plus proche lorsque le point homologue n'existe pas. Ainsi, par itérations successives, elle va déterminer la consolidation qui minimise la distance entre les deux nuages. Il est généralement d'usage de sélectionner au préalable trois points homologues entre les nuages afin de limiter le nombre d'itérations, ce qui en fait plutôt une technique semi-automatique. La seconde méthode consiste à rechercher des entités géométriques homologues et non des points. Ces entités sont de type sphères, cylindres, plans ou droites mais leur extraction allonge le temps de traitement.

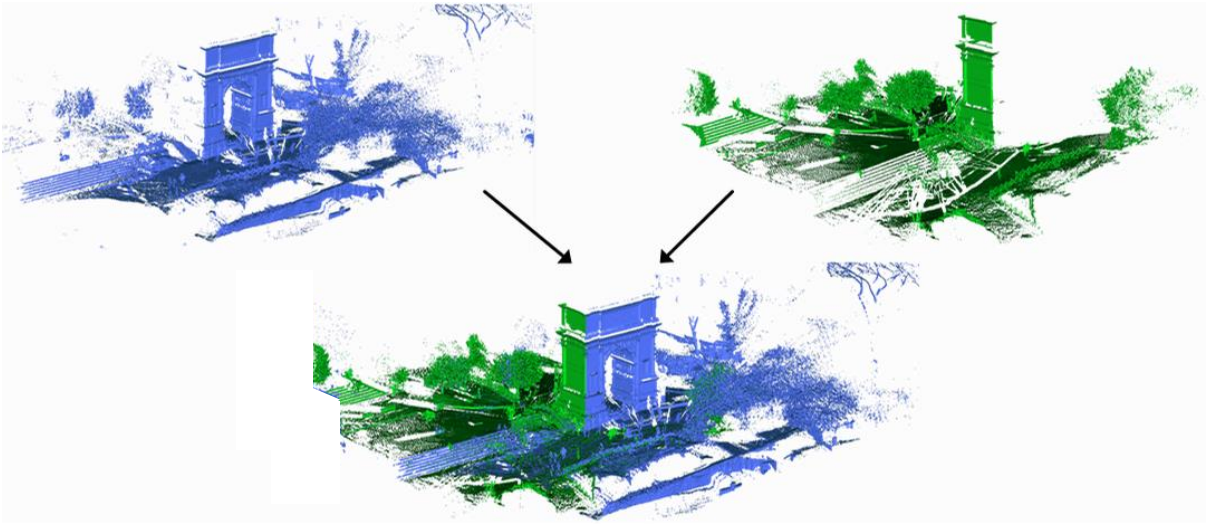


Figure 2 : Consolidation de deux scans en un nuage de points unique (Theiler et al., 2015)

De cette consolidation résulte un seul nuage de points regroupant la totalité des points des différents scans dans un système unique. Pour certaines applications, le rattachement n'est pas essentiel et la consolidation suffit. Néanmoins, il est parfois nécessaire de géoréférencer le nuage de points dans un système défini. Le rattachement peut alors se faire en mesurant des cibles (ou des points particuliers) à la station totale afin de déterminer leurs coordonnées dans le système désiré. Connaissant les coordonnées de minimum trois points dans les deux systèmes, les 7 paramètres de la transformation d'Helmert (mise à l'échelle, rotations en X, Y et Z et translations en X, Y et Z) peuvent être calculés. La transformation est alors appliquée à l'ensemble du nuage pour le géoréférencer.

2.1.2.3. Nettoyage et échantillonnage

Une autre étape est essentielle à l'obtention d'un nuage de points cohérent : le nettoyage. Celui-ci comprend l'élimination, ou du moins l'atténuation, du bruit et des points aberrants (*outliers*) ou indésirables. L'échantillonnage du nuage peut également être nécessaire.

Tout d'abord, la présence de bruit et d'*outliers* peut affecter la qualité de la segmentation, par exemple en lissant des éléments au départ nets, cassants (Nurunnabi *et al.*, 2016). Ces points aberrants ne devraient pas être présents dans le nuage puisqu'ils ne correspondent pas à des mesures d'éléments de la scène. En effet, ce sont des points qui proviennent notamment d'une mauvaise réflexion sur les surfaces particulières de certains objets, de conditions météorologiques désavantageuses (pluie, vent) ou du bruit inhérent à l'appareil de mesure (Van Genechten, 2008 ; Han *et al.*, 2017). Plusieurs algorithmes de filtrage ayant pour but d'éliminer ce bruit existent. Han *et al.* (2017) les distinguent en sept catégories parmi lesquelles se trouvent des techniques de filtrage basées sur les statistiques, sur le voisinage des points ou encore sur le traitement du signal.

Outre le bruit inhérent aux mesures, Landes *et al.* (2011) mettent en évidence la présence de points non-désirables pour le rendu final. Ces points proviennent par exemple d'obstacles, fixes ou mobiles (véhicules, piétons, arbres), situés entre le scanner et la zone mesurée. La présence d'ouvertures transparentes (portes et fenêtres) sur les bâtiments engendre également des points supplémentaires « inutiles » puisque le faisceau laser va pénétrer à l'intérieur des pièces jusqu'à atteindre une surface opaque.
De plus, le champ de vision de la plupart des scanners laser terrestres est défini par défaut à 360° pour la direction horizontale et aux alentours de – 60° à 90° selon la verticale. Hors, sur terrain, il n'est pas toujours utile de mesurer une telle étendue de points autour de la position

du scanner. En effet, si l'objectif consiste juste à mesurer un bâtiment en trois dimensions, les points levés à l'opposé du bâtiment par rapport à la station ne sont d'aucune utilité, ils augmentent juste le volume des données. Pour éviter cela, il est généralement possible de modifier les paramètres du champ de vision du scanner utilisé. Afin de mesurer uniquement la zone d'intérêt, l'utilisateur peut fixer les angles horizontaux et verticaux de son choix (Leica Geosystems, 2016).

Le sous-échantillonnage du nuage de points poursuit deux objectifs : homogénéiser la densité du nuage et réduire le volume des données. La variabilité dans la densité des points a deux origines. La première vient de la disposition de la scène, c'est-à-dire l'orientation et la distance des surfaces levées par rapport au scanner lors de la prise de mesures (Nguyen & Le, 2013). La seconde découle de la nécessité de multiplier les scans depuis différentes positions pour mesurer la totalité de l'objet ou du secteur étudié. Les zones de recouvrement entre ces nuages, mesurées depuis au moins deux stations, sont alors nettement plus denses et alourdissent le nuage total. L'homogénéisation peut se faire en échantillonnant le nuage sur base d'une densité de points fixes (Landes *et al.*, 2011). Une autre manière très simple de réduire la taille des données est de conserver un point tous les X points. L'inconvénient de cette méthode réside néanmoins dans le fait qu'une telle réduction du nombre de points peut engendrer une perte d'informations précieuses pour des éléments de petite taille ou de forme particulière. Il est également possible de procéder à un échantillonnage aléatoire en fixant simplement le nombre désiré de points. Des techniques particulières de sous-échantillonnage ont été élaborées afin de conserver une densité de points plus importante lorsque les éléments présentent une courbure élevée alors que celle-ci est diminuée dans les zones représentables avec peu de points (Van Genechten, 2008).

2.1.2.4. Segmentation et classification

La segmentation et la classification sont deux étapes essentielles pour extraire de l'information des nuages de points. Elles sont parfois regroupées en une seule étape selon la méthode utilisée. La segmentation est définie comme la subdivision d'un nuage en un ensemble de groupes de points ayant des propriétés similaires, appelés segments (Landes *et al.*, 2011 ; Nguyen & Le, 2013 ; Poux *et al.*, 2014 ; Che & Olsen, 2018). La classification est quant à elle caractérisée par Grilli *et al.* (2017) comme l'attribution d'une classe spécifique (*label*) à ces ensembles de points.

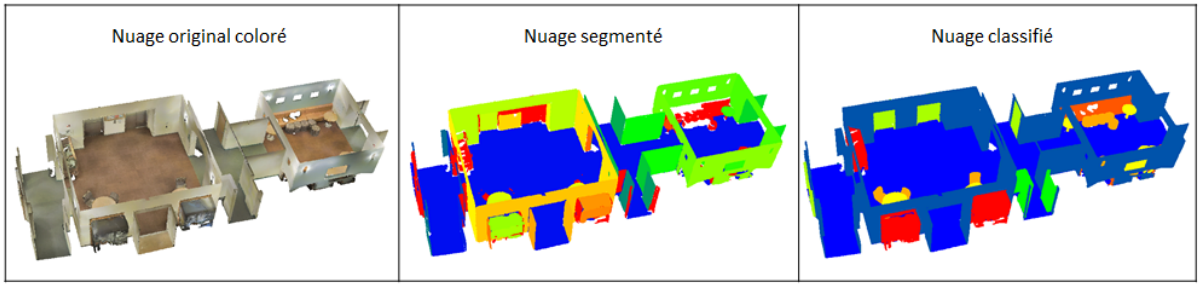


Figure 3 : Exemple de segmentation et classification d'un nuage de points

Il existe de nombreuses possibilités pour segmenter un nuage de points. Tout d'abord, la segmentation peut être réalisée manuellement. Celle-ci étant cependant extrêmement chronophage, d'autant plus que la quantité de points traités augmente, une série de méthodes de segmentations automatiques ou semi-automatiques ont été développées (Poux *et al.*, 2014). Celles-ci peuvent être regroupées en plusieurs catégories selon leurs caractéristiques. Ces catégories peuvent différer un peu d'un auteur à l'autre. En effet, Nguyen & Le (2013) distinguent les cinq classes suivantes : les méthodes basées sur la détection de coins/contours, celles sur la création de régions, celles fondées sur le calcul d'attributs, celles basées sur l'ajustement de modèles/primitives géométriques ou encore les méthodes considérant les nuages de points comme des graphes. Poux *et al.* (2014) regroupent les différentes méthodes en quatre catégories : segmentation par détection de coins/contours, par reconnaissance de formes, segmentation basée sur les surfaces et enfin les segmentations « hybrides », c'est-à-dire celles qui combinent plusieurs méthodes. Grilli *et al.* (2017) ajoutent à ces quatre catégories une cinquième reprenant la segmentation par *machine learning*. Ces cinq méthodes de segmentation sont présentées brièvement ci-dessous avec leurs avantages et leurs inconvénients.

La segmentation par détection de contours s'effectue en deux étapes principales. La première consiste à détecter les arêtes pour décrire les limites/bords des différentes régions. La seconde génère les segments finaux en regroupant les points compris entre ces limites. Les arêtes sont définies par les points où un changement excédant un certain seuil est observé dans les propriétés des surfaces locales. Les propriétés de surfaces les plus couramment utilisées sont les normales, les gradients, la courbure principale ou d'autres dérivées (Poux *et al.*, 2014 ; Grilli *et al.*, 2017 ; Xie *et al.*, 2020). L'avantage principal de ce type d'algorithme est la rapidité de la segmentation. Cependant, ils sont très sensibles à la présence de bruit et de densités inégales dans les nuages de points ce qui peut conduire à des résultats peu fiables (Nguyen & Le, 2013 ; Grilli *et al.*, 2017).

La segmentation par reconnaissance de formes, également connue en tant que segmentation basée sur l'ajustement de modèles, se base sur le principe que de nombreux objets artificiels sont décomposables en primitives géométriques telles que plans, sphères, cylindres ou encore cônes. Dès lors, ces méthodes ont pour objectif de détecter et extraire ces formes géométriques (segments) dans un nuage de points. Deux algorithmes sont généralement utilisés pour développer ce type de segmentation : la transformée de Hough et l'algorithme RANSAC (*RANdom SAmple Consensus*) (Nguyen & Le, 2013 ; Poux *et al.*, 2014 ; Grilli *et al.*, 2017 ; Xie *et al.*, 2020).

La transformée de Hough a initialement été élaborée pour la détection de lignes en imagerie 2D (Hough, 1962). Elle a ensuite été adaptée aux nuages de points 3D afin d'y détecter les surfaces planes (Vosselman *et al.*, 2004) ou encore les sphères et les cylindres.

L'algorithme RANSAC (Fischler & Bolles, 1981) est la méthode la plus connue pour ce type de segmentation. Initialement développé pour l'analyse 2D, il est désormais largement utilisé pour détecter les primitives géométriques dans les nuages de points. Son principe de fonctionnement est expliqué par Schnabel *et al.* (2007). Il consiste à extraire des formes géométriques du nuage en sélectionnant aléatoirement un ensemble de points, correspondant au nombre de points minimal pour décrire de manière unique une forme donnée, et en construisant la primitive associée à ces points. La primitive candidate est alors testée sur le reste des données afin de déterminer combien de points sont bien approximés par celle-ci et lui attribuer un score sur cette base. Un certain nombre d'essais sont effectués et la forme obtenant le plus grand score, c'est-à-dire celle approximant le plus de points, est extraite du nuage. L'algorithme continue ensuite sur les données restantes.

Ces deux algorithmes sont robustes face au bruit et aux *outliers* mais ils nécessitent des temps de traitement élevés, consomment beaucoup de mémoire et leur efficacité peut être limitée par la complexité des formes rencontrées (Schnabel *et al.*, 2007 ; Nguyen & Le, 2013 ; Grilli *et al.*, 2017).

La troisième catégorie reprend les méthodes de segmentation basée sur les surfaces comprenant les algorithmes de croissance de région. Ceux-ci se basent sur l'information de voisinage pour regrouper les points spatialement proches ayant des propriétés surfaciques similaires et ainsi construire différentes régions (segments) (Nguyen & Le, 2013 ; Poux *et al.*, 2014). La croissance de région se fait en deux étapes principales : tout d'abord la sélection d'une ou plusieurs graines et ensuite la croissance à partir de ces graines sur base de critères de similarité prédéfinis pour former une région isolée (Grilli *et al.*, 2017). Le principe de

l'algorithme est illustré à la figure 4. Ces méthodes sont moins sensibles au bruit que celles par détection de contours mais les temps de traitement sont relativement longs et la qualité de la segmentation dépend du choix des graines et des critères de similarité choisis (Nguyen & Le, 2013).

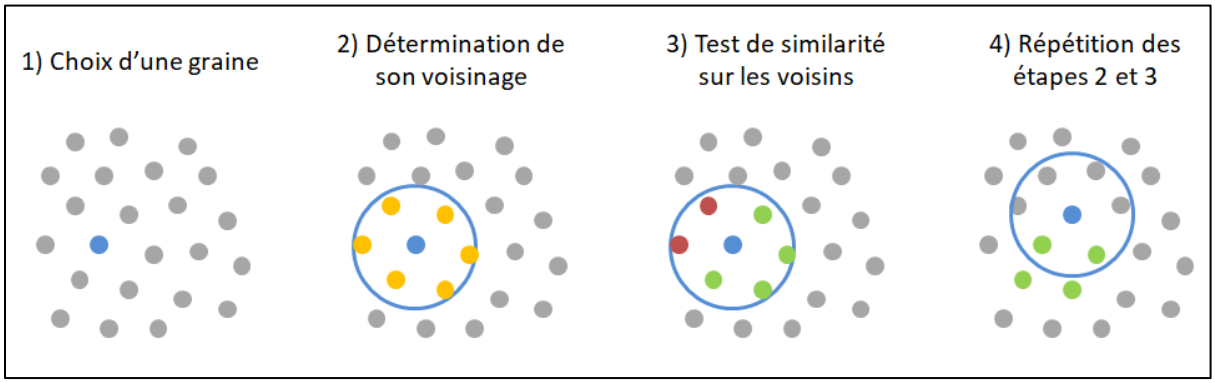


Figure 4 : Principe de la croissance de région

Les segmentations hybrides combinent plusieurs méthodes afin de profiter des avantages de chaque méthode tout en évitant ou limitant leurs points faibles (Grilli *et al.*, 2017).

La dernière catégorie présentée par Grilli *et al.* (2017) comprend les segmentations par *machine learning*. Celles-ci utilisent l'intelligence artificielle pour effectuer des choix de segmentation à partir de données d'entrainement. Trois types d'apprentissages sont distingués : supervisé, non-supervisé et par renforcement (Poux *et al.*, 2016). Le premier est guidé par l'apport de données d'entrainement étiquetées, c'est-à-dire que les classes sont définies au préalable. A l'inverse, l'apprentissage non-supervisé se fait de manière autonome pour détecter la structure des données sans définir de résultat attendu. L'apprentissage par renforcement consiste à récompenser ou pénaliser les choix faits afin d'optimiser la prise de décision. Ces méthodes par *machine learning* sont robustes et fournissent de meilleurs résultats, surtout pour des scènes complexes où les objets sont difficilement associables à des primitives géométriques, mais elles requièrent de longs temps de calcul (Nguyen & Le, 2013).

De nombreux algorithmes ont été développés pour mettre en œuvre ces différentes méthodes de segmentation en essayant de palier à leurs défauts tout en utilisant leurs avantages. La partie 2.2 présente une partie de ceux-ci.

La classification, définie précédemment comme l'attribution d'une classe aux points du nuage, peut également être divisée en plusieurs catégories selon la méthode employée. Che *et*

al. (2019) distinguent trois catégories sur base du type de données utilisées en entrée : la classification par points, celle par segments ou celle par objets. La méthode par points consiste à analyser et associer une classe à chaque point individuellement, elle n'inclut par conséquent pas d'étape de segmentation. La classification par segments, comme son nom l'indique, requiert une segmentation préalable puisqu'une même classe est attribuée aux points d'un même segment. La classification par objets intervient également après une étape de segmentation. Néanmoins, les segments appartenant au même élément sont d'abord regroupés ensemble pour former un objet auquel une classe unique est alors associée.

2.1.2.5. Modélisation

Même si le nuage de points classifié peut être utilisé tel quel pour visualiser ou extraire de l'information, une étape supplémentaire permet d'obtenir un « tout » en liant les points du nuage : c'est la modélisation. Landes *et al.* (2011) distinguent trois types de modèles :

1) Le modèle « reconstruit » : il s'agit du modèle couramment utilisé en archéologie puisqu'il est élaboré à partir de relevés et qu'il est complété sur base de connaissances d'experts afin de reconstituer les parties manquantes, c'est-à-dire les parties ne disposant pas de données.
2) Le modèle « tel que saisi » : l'objet est reconstitué fidèlement dans son état au moment de l'acquisition, exclusivement sur base des mesures effectuées. C'est le type de modèle pouvant être réalisé par un géomètre.
3) Le modèle « tel que construit » : il consiste à représenter l'objet tel qu'il a été construit, avant d'éventuelles dégradations, à partir de connaissances fiables sur l'objet (plans par exemple) et de mesures additionnelles.

Deux techniques de modélisation sont fréquemment utilisées : la modélisation par maillage et la modélisation géométrique (Landes *et al.*, 2011).

La première technique consiste à relier les points d'une même entité entre eux pour générer une surface. Sur base d'une triangulation de Delaunay par exemple, des facettes sont construites entre les différents points. Le maillage permet ainsi d'obtenir un modèle fidèle à la réalité dont l'inconvénient principal est la lourdeur du fichier résultant. En effet, les géométries simples telles que des plans sont représentées de la même façon que les surfaces complexes alors qu'elles pourraient être modélisées plus légèrement.

La modélisation géométrique consiste quant à elle à représenter les entités du nuage de points à l'aide de primitives géométriques décrites par quelques paramètres. Il en résulte un fichier allégé par rapport au modèle maillé dans lequel les objets correspondent à des formes géométriques. Cette technique renvoie plutôt à la catégorie des modèles « tels que construits ».

2.2. Méthodes de segmentation et classification

La segmentation et la classification des nuages de points ne sont pas des tâches aisées puisqu'il s'agit de données complexes et variées (Nguyen & Le, 2013 ; Vo *et al.*, 2015 ; Grilli *et al.*, 2017). De nombreuses recherches sont dès lors consacrées à ce sujet afin d'améliorer et accélérer le processus. Certaines d'entre elles sont détaillées ci-dessous.

2.2.1. Algorithmes de segmentation

La qualité de la segmentation est généralement impactée par la présence de bruit et de points aberrants. Afin de faire face à ce problème, Nurunnabi *et al.* (2016) proposent une croissance de région basée sur une estimation robuste des caractéristiques géométriques locales telles que la normale et la courbure. Celles-ci sont couramment obtenues à partir d'une analyse en composantes principales (ACP), considérée non-robuste. En combinant cette dernière avec un algorithme de maximisation de la cohérence et de minimisation des distances, les caractéristiques géométriques obtenues sont plus robustes. L'algorithme de segmentation résultant permet une segmentation plus précise et plus résistante au bruit que RANSAC pour un temps de traitement similaire.

Che & Olsen (2018) ont développé une méthode de segmentation en deux étapes dédiée aux données provenant d'un laserscan terrestre. Leur méthode propose une alternative à l'estimation des normales en chaque point, couramment utilisée pour la croissance de région, qu'ils qualifient de sujette à erreur. Tout d'abord, les points d'intersections et de bordures des surfaces sont extraits grâce à « Norvana » (*NORmal VARiation ANAlysis*). Une partie du bruit est également éliminée lors de cette étape. Ensuite, une croissance de région est appliquée sur les points des surfaces lisses. Celle-ci est dès lors plus robuste puisque les points problématiques (coins et bruit) ont été retirés. Ces deux étapes se basent sur une structure angulaire grillagée, utilisée pour stocker les points d'un scan et adaptée par les auteurs pour prendre en compte plusieurs scans à l'aide d'une indexation. L'algorithme permet ainsi une

segmentation rapide de millions de points mais il est uniquement valable pour les données structurées provenant d'un laserscan terrestre.

La segmentation est un processus lourd et coûteux en temps puisqu'il nécessite le traitement de millions de points. La voxelisation, méthode de subdivision de l'espace en voxels (cubes) de plus ou moins petite taille, permet de définir plus efficacement le voisinage des points et de réduire les temps de traitements. Par conséquent, plusieurs algorithmes de segmentation ont été mis en œuvre sur base de ce principe.

Par exemple, Vo *et al.* (2015) présentent un algorithme de croissance de région basé sur un *octree* en deux phases. La première consiste à diviser le nuage de points en voxels et appliquer une croissance de région sur ces voxels dans leur totalité. Les voxels adjacents ayant des caractéristiques similaires sont ainsi regrouper en segments qualifiés de grossiers. La deuxième phase consiste à affiner la segmentation en considérant les points à l'intérieur des voxels qui n'ont pas encore été assignés à un segment car ils présentent des caractéristiques distinctes de leurs voisins. Cette méthode offre de bons résultats avec un temps de traitement réduit grâce à la voxelisation. Celle-ci permet de simplifier les données de départ, de les indexer et de définir rapidement le voisinage.

La subdivision du nuage de points en voxels est également utilisée dans la méthode proposée par Poux & Billen (2019). Celle-ci se base sur l'extraction de deux jeux de caractéristiques. Le premier reprend les attributs liés à la forme (SF1) déterminés à partir des coordonnées X, Y et Z des points. Le deuxième type d'attributs est quant à lui lié à la connectivité entre voxels (SF2). Le nuage est ensuite structuré en un set d'éléments connectés (CEL) à partir de ces attributs. Ils exposent ensuite un arbre décisionnel fondé sur les connaissances pour permettre la classification liée à l'infrastructure. Les auteurs comparent ensuite les résultats de leur méthode avec des méthodes efficaces de type *deep learning* telles que le réseau « PointNet » proposé par Qi *et al.* (2017). La méthode proposée se montre robuste au bruit, aux occlusions et aux irrégularités et elle est performante pour la détection et la classification des éléments structurels (sol, plafond et mur).

Le concept de voxels est aussi présent dans la méthode de segmentation spécifique aux plans de Dong *et al.* (2018). Afin de pallier aux limitations récurrentes dans la segmentation de plans (difficultés pour détecter les petits plans, bordures incomplètes, etc.), ils proposent une méthode en trois étapes. La première étape consiste, après avoir déterminé les caractéristiques géométriques des points, à diviser le nuage en voxels d'une certaine taille et à classer chacun

d'entre eux en tant que voxel plan ou non plan. Les voxels non plans sont alors divisés à une échelle plus petite. L'algorithme se répète jusqu'à ce que tous les voxels soient classés en tant que plans ou que l'échelle minimale fixée au départ soit atteinte. Ce processus permet de réduire le nombre d'unités à partir desquelles une croissance de région hydride est appliquée, c'est la seconde étape. La dernière étape consiste à améliorer et affiner la segmentation des plans selon le principe d'optimisation énergétique globale. Même s'il s'avère que la méthode présentée est plus efficace que celle de Vo *et al.* (2015) notamment, elle se limite à la segmentation des plans.

Zolanvari *et al.* (2018) comparent également leur travail avec celui proposé par Vo et al. (2015) pour mettre en évidence la rapidité de leur méthode, sans intervention manuelle ou information a priori, pour un résultat équivalent en qualité. Ils introduisent une *Improved Slicing Method* (ISM) afin d'automatiser la segmentation de bâtiments en 3 dimensions (pas uniquement leur façade). L'ISM est une extension de la *Slicing Method*, présentée par Zolanvari & Laefer (2016) et utilisée pour segmenter et détecter les ouvertures d'une façade. Le principe général consiste à diviser les surfaces planes, détectées rapidement avec RANSAC pour éliminer les points externes (végétation, objets intérieurs, etc.), en tranches horizontales identiques selon l'axe principal du bâtiment. Sur base des espaces vides présents dans chaque tranche du nuage, les points constituant les limites des ouvertures et de la façade peuvent être détectés et se voient attribués aux différentes parties du bâtiment (façade avant, arrière ou bien côtés). Les points d'une même partie sont ensuite regroupés pour segmenter la totalité du bâtiment. L'algorithme est capable de détecter les ouvertures dans les toits ainsi que les contours d'éléments plus complexes tels que les cheminées de manière complétement automatique.

2.2.2. Intégration de données topographiques dans le processus

Quelques travaux mettent en évidence l'intérêt d'intégrer des données topographiques pour faciliter ou accélérer le processus de segmentation et de classification d'un nuage de points. Les données topographiques utilisées peuvent provenir de différentes sources telles que des plans 2D (Lesecque, 2019), des cartes topographiques (Oude Elberink, 2020) ou bien des levés topographiques (Djemâ, 2018).

Lesecque (2019) s'intéresse à l'intégration de données sémantiques et spatiales provenant de plans bidimensionnels CAD (*Computer Aided Design*) pour la segmentation et la

classification d'environnements intérieurs. L'objectif de son travail est de développer une méthode de segmentation plus rapide tout en étant aussi précise et exacte que la méthode manuelle. La méthode proposée consiste tout d'abord à extraire les informations sémantiques et spatiales contenues dans le plan pour pouvoir ensuite réaliser la segmentation. Celle-ci est basée sur la création de zones tampons pour chaque élément du plan à partir des informations extraites et sur la détermination de statistiques sur les hauteurs des points des différentes classes. Les résultats obtenus montrent une segmentation plus rapide, peu ou pas influencée par la présence d'occlusions ou de bruit, mais moins exacte que la segmentation manuelle. Certains problèmes affectant la qualité des résultats ont pu être mis en évidence tels qu'un géoréférencement imparfait, causé par de légers décalages entre le plan et le nuage de points, ou encore l'absence d'informations précises sur les hauteurs des éléments puisque le plan est en deux dimensions.

Oude Elberink (2020) présente un travail consacré à la fusion de données issues d'un scanner laser mobile (MLS) avec des cartes topographiques à grande échelle. L'objectif de son travail est d'utiliser l'information sémantique et spatiale contenue dans une carte topographique pour faciliter la classification du nuage de points de densité et apparence variable. La méthodologie développée peut être divisée en trois étapes principales. La première étape reprend les prétraitements effectués sur le nuage de points, c'est-à-dire l'élimination des *outliers* et la distinction entre les points appartenant au sol et les autres. La seconde concerne le traitement des entités polygonales reprises sur la carte topographique (bâtiments, routes, etc.). Sur base des classes prédéfinies et de l'apparence de l'objet, une série de conditions sont appliquées pour guider le processus de segmentation. La dernière étape se consacre aux éléments ponctuels (arbres, lampadaires, etc.) afin de regrouper les points restants du nuage qui appartiennent au même objet. Cette segmentation est adaptée en fonction du type d'objet grâce à l'information fournie par la carte topographique. Une attention particulière est accordée aux entités ponctuelles proches les unes des autres pour associer correctement les points à l'entité correspondante et pas une autre. L'algorithme développé doit également tenir compte de l'imprécision de positionnement sur les données, allant jusqu'à un mètre pour les données MLS. Après avoir classifié les entités polygonales et ponctuelles, les points non assignés sont analysés afin de repérer les points correspondant aux voitures ou piétons et les points qui pourraient faire partie d'un objet non-repris sur la carte. Les entités de la carte topographique qui ne se retrouvent pas dans le nuage de points peuvent également être repérées. Ainsi, outre un apport pour la classification des nuages de points, ce travail met en

évidence l'opportunité de mettre à jour une carte topographique existante sur base des changements détectés à l'aide de mesures effectuées au scanner laser mobile.

Il semblerait que, dans la littérature, peu de recherches aient été menées sur l'intérêt de récolter des informations particulières lors de l'acquisition d'un nuage de points afin de faciliter son traitement. Néanmoins, Djemâ (2018) met en évidence l'intérêt de combiner des mesures issues d'un levé topographique avec celles obtenues par scanner laser afin de développer une méthode de segmentation automatique simple et efficace. Son travail se concentre sur la segmentation des arbres et compare les résultats de trois méthodes différentes, utilisant chacune une quantité croissante de données recueillies sur terrain. La première méthode de segmentation se base uniquement sur la position du centre des arbres ainsi que sur les dimensions moyennes d'arbres, sans distinguer les espèces. La seconde méthode intègre le type d'arbres (feuillu, conifère ou buisson) ce qui permet d'adapter la forme de la primitive géométrique. Enfin, la dernière méthode présentée inclut également le rayon du tronc et une estimation du rayon de la couronne pour affiner les dimensions des primitives géométriques utilisées pour la segmentation. La première méthode n'est pas concluante puisque la qualité de la segmentation est significativement moindre que celle d'une segmentation manuelle pour un temps d'exécution important. Les deux autres techniques fournissent quant à elles des résultats satisfaisants, tant au niveau de la qualité que du temps de traitement.

3. HYPOTHÈSES DE RECHERCHE

L'état de l'art a permis de mettre en évidence l'intérêt porté à la segmentation et à la classification des nuages de points afin d'en tirer de l'information utile. La segmentation manuelle étant longue et fastidieuse, de nombreux algorithmes ont été développés afin d'automatiser le processus. Il s'avère néanmoins que les recherches sont généralement portées sur le post-traitement des données et non sur leur acquisition. Il semble donc intéressant d'introduire une nouvelle approche basée sur une acquisition réfléchie, visant à faciliter le traitement du nuage de points par la suite.

La question à la base de ce travail est la suivante : « Est-il possible d'utiliser une géocodification lors de l'acquisition d'un nuage de points par laserscan pour améliorer le processus de segmentation et classification ? ».

Deux hypothèses peuvent être formulées à partir de cette question :

1) Il est effectivement possible de créer une géocodification pour l'acquisition d'un nuage de points au laserscan et celle-ci permet d'adapter le processus de segmentation/classification aux éléments présents sur le terrain.

2) Cette méthode de segmentation/classification est plus rapide qu'une segmentation manuelle pour des résultats équivalents en précision.

Pour vérifier ces hypothèses, deux méthodes différentes ont été mises en place. Elles sont détaillées dans le chapitre suivant.

4. MÉTHODOLOGIE

Ce chapitre est consacré à la présentation de la méthodologie mise en œuvre pour la réalisation de ce travail. Il s'agit d'une méthodologie générale, reproductible dans d'autres conditions de travail. Deux méthodes distinctes ont été développées. La première utilise la géocodification traditionnelle d'un levé topographique, combinée ensuite avec un nuage de points afin de le classifier. La seconde méthode est plus innovante puisque la géocodification consiste à utiliser des cibles dimensionnées et positionnées spécifiquement pour permettre une segmentation/classification guidée du nuage de points. Les étapes pour implémenter ces deux méthodes sont néanmoins semblables. Le diagramme à la figure 5 résume la procédure suivie. Celle-ci commence par l'établissement de la géocodification, adaptée à chaque méthode proposée, suivie de l'acquisition des données sur terrain. Plusieurs prétraitements sont ensuite nécessaires pour pouvoir exploiter ces données tels que la consolidation et le nettoyage du nuage de points ou encore la détermination des coordonnées des cibles ou des points levés. Après avoir préparé les données, la segmentation et la classification du nuage peuvent avoir lieu. La dernière étape consiste à analyser et valider les résultats sur base d'une vérité terrain. La méthode des cibles, en vert sur la figure, est présentée en premier puisque les traitements appliqués au nuage de points sont également valables pour la seconde méthode, en rouge. Le jaune illustre les données ou résultats communs aux deux méthodes.

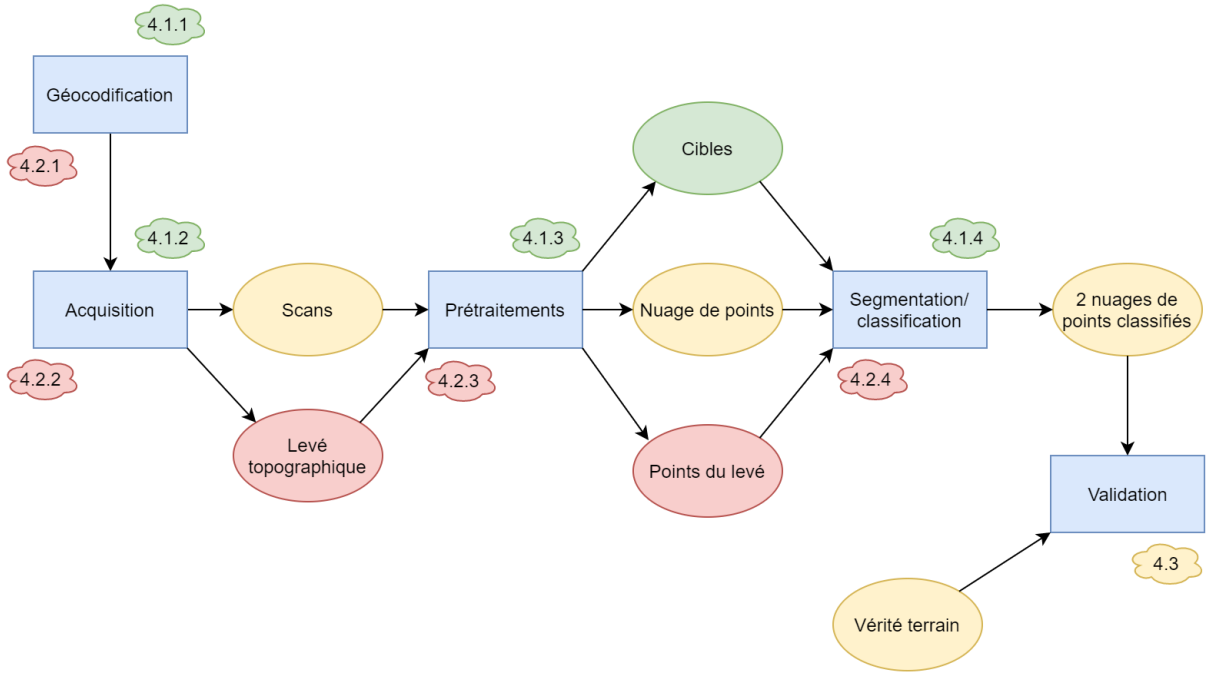


Figure 5 : Schéma résumant la méthodologie

4.1. Méthode des cibles

La méthode des cibles consiste, comme son nom l'indique, à effectuer la segmentation et la classification du nuage de points sur base de cibles positionnées de manière spécifique dans la zone de travail. A chaque cible est associé un code permettant d'assigner directement l'objet à sa classe, c'est le principe de la géocodification. Les différentes cibles sont dimensionnées et positionnées afin de correspondre au mieux à l'objet et de faciliter le processus de segmentation. Les différentes étapes de cette méthode sont détaillées ci-dessous, depuis la géocodification jusqu'à la segmentation/classification.

4.1.1. Géocodification

En tant que première étape du processus, la géocodification doit être établie en tenant compte des éléments présents sur le terrain et plus particulièrement ceux présentant un intérêt pour l'objectif du travail. Dans le cadre de ce mémoire, nous nous sommes intéressés au bâti avec l'objectif de distinguer les murs, les portes, les fenêtres et éventuellement d'autres éléments tels que les gouttières. Il peut également être intéressant de différencier la toiture du reste du bâtiment mais elle n'est pas toujours visible par laserscan terrestre. Il est néanmoins possible d'adapter la géocodification à toute sorte d'objets comme des lampadaires, tuyaux, etc.

A chaque classe d'objets est associé un code spécifique sous forme d'un numéro. Ce numéro est apposé au-dessus d'une cible qui sera positionnée sur l'objet correspondant. En outre, la cible est dimensionnée de manière spécifique selon le type d'objet. A titre d'exemple, les surfaces d'un mur et d'une gouttière sont fort différentes. Les cibles destinées aux gouttières sont donc plus petites afin de ne pas cacher l'objet et de mieux coller à sa géométrie. Il faut néanmoins faire attention à ne pas sous-dimensionner les cibles vis-à-vis de la résolution du laserscan. Elles doivent être reconnaissables pour être utilisées dans la suite des traitements.

C'est également à ce stade du travail qu'il faut réfléchir au positionnement des cibles et à leur nombre pour permettre une classification efficace du nuage. Cette réflexion a cependant été faite uniquement en amont de l'acquisition. La géocodification aurait sans doute pu être améliorée pour certaines classes en prenant en compte les difficultés rencontrées lors de la création de l'algorithme. Une deuxième acquisition n'était toutefois pas possible. Ce point sera discuté à la fin de ce travail (cf. 7. Discussion).

En ce qui concerne les murs, la géocodification se limite à l'utilisation d'une cible par mur, positionnée en bas à gauche de celui-ci et numérotée avec le code correspondant.
Une cible est également attribuée à chaque porte devant être extraite. Elle est placée relativement bas afin de pouvoir faire la distinction avec les fenêtres sur base de la position. Deux cibles suffisent pour extraire toutes les fenêtres d'une même façade. En effet, celles-ci sont généralement alignées dans un même plan ce qui rend inutile l'utilisation d'une cible par fenêtre. Les cibles sont dès lors placées sur les deux fenêtres situées aux extrémités de la façade. Pour les deux types d'ouvertures, les cibles sont positionnées sur la vitre ce qui permet, si nécessaire, de facilement retirer les points du nuage correspondant à ces cibles.
Enfin, une cible de plus petite taille est prévue pour chaque gouttière. Sa position importe peu tant qu'elle suit bien la forme de l'objet et ne modifie pas ses caractéristiques géométriques. La figure ci-dessous illustre la géocodification utilisée pour cette méthode.

Figure 6 : Positionnement des cibles pour la méthode des cibles

4.1.2. Acquisition

L'acquisition des données nécessaires pour la segmentation constitue la deuxième étape du processus. Après avoir numéroté et imprimé des cibles, celles-ci sont positionnées sur tous les objets à traiter en suivant la méthodologie détaillée dans la partie géocodification. Ces cibles présentent deux utilités puisque, en plus de leur rôle principal pour la segmentation, elles serviront pour l'assemblage des différents scans en post-traitement. Toutefois, afin d'assurer un nombre suffisant de cibles entre deux scans successifs (cinq cibles communes sont recommandées), des cibles supplémentaires sont réparties dans la zone au fur et à mesure des mesures. Parmi ces cibles supplémentaires, certaines sont positionnées à la verticale de clous présents dans la zone. Ces clous matérialisent la polygonale établie pour la

seconde méthode (cf. 4.2.2.) et permettent de géoréférencer les deux jeux de données dans le même système de référence (toujours en post-traitement). Dans le cadre de cette méthode, un système local aurait suffi pour traiter les données. Néanmoins, pour travailler sur le même jeu de données et pouvoir comparer les deux méthodes développées dans ce travail, il était nécessaire de géoréférencer l'ensemble des mesures dans un système prédéfini.

Une fois les cibles positionnées, l'acquisition au laserscan terrestre commence. Comme la durée d'un scan dépend de la résolution fixée, il faut choisir le bon compromis entre temps et qualité. En effet, une résolution plus fine assure la reconnaissance de cibles positionnées loin de la station mais elle engendre une durée d'acquisition plus importante. Une résolution différente peut toutefois être appliquée d'un scan à l'autre pour correspondre au mieux à la situation. Plusieurs stations successives sont nécessaires pour couvrir la zone étudiée. La position de ces stations doit être réfléchie afin de minimiser les occlusions et effectuer une acquisition complète.

4.1.3. Prétraitements

A l'issue de l'acquisition, les données récoltées sont une série de scans individuels, chacun initialement référencé dans un système relatif à la position du scanner. Pour pouvoir utiliser ces données dans l'algorithme de segmentation, plusieurs traitements sont nécessaires. Les différents concepts associés à ces traitements ont été détaillés précédemment (cf. 2.1.2.).

Le premier prétraitement consiste à géoréférencer les nuages individuels. Il s'agit ici d'un géoréférencement indirect, précédé par une consolidation (assemblage des scans). La première étape consiste à importer les différents scans et à éliminer, si le cas se présente, ceux qui ne seraient pas utiles. Ensuite, il faut vérifier que les cibles repérées par le logiciel utilisé sont effectivement des cibles. Dans le cas contraire, il faut corriger ces artéfacts puisqu'ils pourraient impacter la suite des traitements. Il faut également vérifier si c'est bien le centre des cibles qui est détecté. Si ce n'est pas le cas, il faut rectifier manuellement la position. Cette étape est importante puisque c'est sur base de ces cibles que la consolidation des nuages individuels est effectuée. La présence d'erreurs dans les cibles entacherait la qualité du nuage total. Le nuage consolidé peut ensuite être rattaché à un système de référence particulier. Pour ce faire, les coordonnées des cibles connues (minimum trois) sont importées. Connaissant les coordonnées de ces points dans les deux systèmes (local et global), les

paramètres de la transformation d'Helmert sont déterminés et appliqués à l'ensemble du nuage.

Le nuage de points est ensuite nettoyé grossièrement de manière à réduire son volume et éliminer une partie du bruit qui pourrait nuire à la segmentation. Pour réduire la taille du nuage, les points éloignés de la zone d'intérêt sont supprimés étant donné qu'ils ne présentent aucune utilité pour la suite. Le bruit causé par le reflet des portes et des fenêtres peut être problématique puisque un mur peut être reflété là où il n'existe pas vraiment. Ce genre de situation pourrait entraîner une certaine confusion dans le processus de segmentation. Il est donc préférable de supprimer ces points au préalable.

L'étape suivante consiste à échantillonner le nuage pour diminuer sa taille et uniformiser la densité de points. En effet, le volume des données acquises par laserscan est généralement conséquent ce qui peut poser problème lors des traitements. Afin de ne pas saturer la mémoire de l'ordinateur, il est habituel de procéder à un sous-échantillonnage. Il faut toutefois faire attention de ne pas supprimer trop de points, cela entrainerait une perte d'informations.

Les traitements appliqués au nuage de points détaillés ci-dessus seront à prendre en compte dans le développement de la deuxième méthode.

Cependant, les prétraitements pour cette méthode-ci ne s'arrêtent pas là. Pour pouvoir utiliser les cibles dans l'algorithme de segmentation, il faut connaître leurs coordonnées ainsi que le code qui leur est associé. Les coordonnées peuvent être extraites directement. En effet, si les cibles ont été correctement repérées avant l'assemblage, les coordonnées X, Y et Z de leur centre sont connues et fournies par le logiciel. Par contre, l'attribution du code n'a pas été faite automatiquement. La reconnaissance des chiffres n'étant pas intégrée, chaque code a été attribué manuellement à la cible correspondante sur base de sa position dans la zone. Les cibles utilisées uniquement pour l'assemblage se voient affecter un code particulier permettant de les distinguer des autres cibles. Un identifiant est également associé à chaque cible.

Au final, deux fichiers résultent de ces prétraitements. Le premier est un fichier contenant la totalité du nuage de points (coordonnées et intensité) qui sera utilisé pour les deux méthodes. Le second est un fichier texte structuré de la manière suivante :

ID ; X ; Y ; Z ; code

A chaque ligne correspond une cible. Ces deux fichiers seront combinés ensemble pour procéder à la segmentation et à la classification du nuage.

4.1.4. Segmentation et classification

L'utilisation d'une géocodification permet d'adapter le processus de segmentation et de classification à chaque type d'objets devant être extraits du nuage de points. Grâce aux coordonnées et au code de chaque cible, la segmentation peut être guidée. Afin d'effectuer la segmentation classe par classe, les cibles sont au préalable triées et séparées sur base de leur code. De cette manière, les cibles correspondant aux murs sont regroupées entre elles, celles associées aux portes également et ainsi de suite. Le fait de procéder classe par classe permet également d'alléger le nuage au fur et à mesure des traitements. En effet, une fois qu'une classe est attribuée à des points, ils sont mis de côté pour la suite de l'algorithme.

L'algorithme a été développé de manière à pouvoir combiner plusieurs méthodes de segmentation telles que la croissance de région ou RANSAC (cf. 2.1.2.4) en fonction de ce qui semble le plus adapté. A cet effet, une série de fonctions ont été développées. L'ordre de succession et les dépendances entre fonctions sont illustrés sur le schéma à la figure 7. Les fonctions principales sont détaillées ci-dessous :

(1) Détermination du voisinage sphérique d'un point. Cette fonction consiste à déterminer les voisins d'un point sur base d'un rayon de recherche définit par l'utilisateur. Tous les points situés à une distance du point inférieure au rayon sont enregistrés comme voisins de ce point.

(2) Analyse en composantes principales (ACP) afin d'extraire les valeurs et vecteurs propres d'un ensemble de points.

(3) Calcul des normales des points. En appliquant l'ACP sur le voisinage d'un point, la normale de ce point peut être extraite puisqu'elle correspond au plus petit des vecteurs propres.

(4) Détermination des paramètres du plan tangent à un point sachant que l'équation d'un plan est $ax + by + cz + d = 0$. Connaissant le vecteur normal à un point (plus petit vecteur propre), les paramètres du plan tangent à ce point peuvent être obtenus. En

effet, les paramètres a, b et c correspondent aux trois composantes du vecteur normal et le paramètre d peut être calculé à partir d'un point du plan.

(5) Attribution des points appartenant à un plan. Sur base des paramètres d'un plan, les points situés à une distance de celui-ci inférieure à un certain seuil peuvent être extraits.

(6) Croissance de région si respect d'un (ou plusieurs) critère(s). A partir d'une graine (point de départ de la croissance de région), les points voisins sont testés. S'ils remplissent la condition fixée, ils sont ajoutés dans la région et deviennent à leur tour une graine pour continuer la croissance.

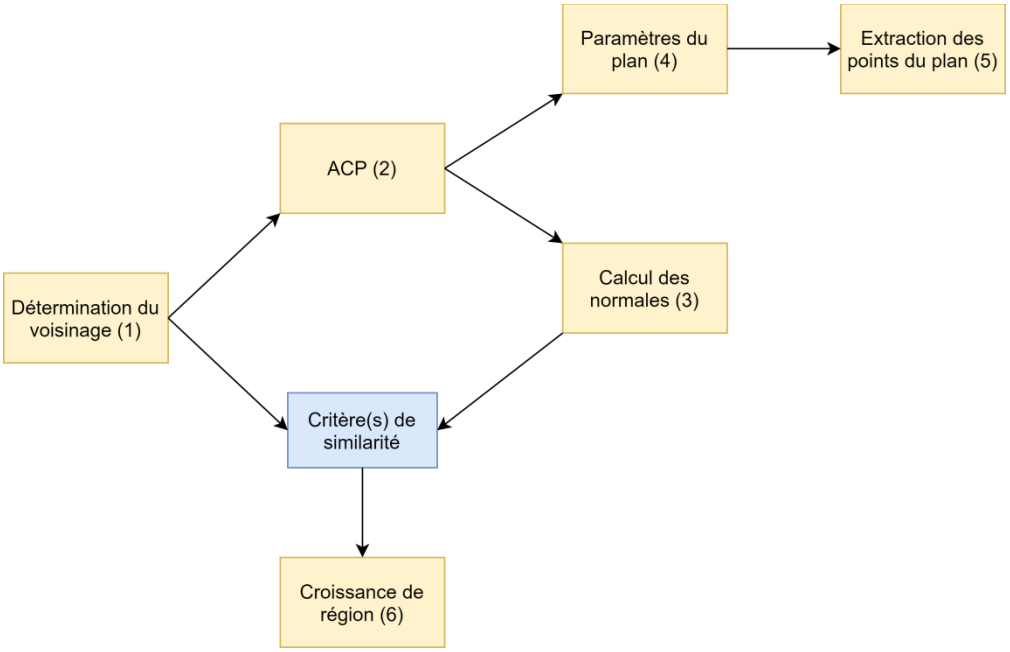


Figure 7 : Schématisation des principales fonctions utilisées pour la segmentation

A partir de ces fonctions diverses, les murs, portes, fenêtres et gouttières peuvent être extraits du nuage de points. La méthodologie employée pour chaque classe est détaillée ci-dessous.

4.1.4.1. Murs

La segmentation des murs se fait en trois étapes principales :

1) Détermination du plan s'ajustant le mieux au mur ;
2) Elimination des points situés au-dessus du mur (végétation, toit,…) ;
3) Elimination des points situés dans le prolongement du mur (sol,…).

Ces trois étapes sont détaillées ci-dessous.

Au départ, l'objectif était d'utiliser les coordonnées de la cible positionnée sur le mur pour extraire, à partir de ces voisins proches, le plan correspondant à ce mur (4). Il s'est néanmoins avéré que le plan extrait de cette manière n'était pas suffisamment précis pour segmenter correctement le mur. Pour affiner la segmentation, l'algorithme RANSAC a été employé pour chaque mur. Le principe de cet algorithme a été expliqué au point 2.1.2.4. Afin de limiter le nombre de plans testés, certaines conditions ont été appliquées pour améliorer et adapter cet algorithme. Tout d'abord, pour ne pas calculer la distance au plan testé (5) depuis tous les points du nuage, seuls les points situés à une certaine distance du plan formé par la cible sont traités. Ensuite, une condition sur la normale du plan testé a été ajoutée : elle doit être proche de la normale du plan initial (associé à la cible) pour que le plan soit pris en compte. Ces deux précautions permettent de limiter le nombre d'itérations à effectuer pour extraire le meilleur plan. Enfin, le plan à tester n'est pas définit par trois points aléatoires comme le fait l'algorithme RANSAC « original ». En considérant les coordonnées de la cible, un point du plan est déjà connu. A chaque itération, ce point peut donc être combiné avec deux points aléatoires pour déterminer l'équation du plan à tester. A la fin des itérations, le plan comprenant le maximum de points est conservé comme mur.

Toutefois, un plan n'étant pas limité en extension (hauteur et longueurs infinies), des points extérieurs au mur se verraient également attribuer cette classe. Pour éliminer les points situés au-dessus du mur et espacés de celui-ci, un histogramme des altitudes des points est établi. Il reprend le nombre de points compris par tranche d'altitude. Si une ou plusieurs tranches intermédiaires ne comprennent aucun point, elles marquent la séparation entre le mur et les points au-dessus. Ainsi, les points dont l'altitude est supérieure à celle de ces tranches vides peuvent être rejetés car ils n'appartiennent pas au mur.

L'analyse se poursuit pour éliminer les points situés dans le prolongement des murs. Pour cela, une grille 2D est établie selon les axes X et Y sur les points extraits, candidats à la classe « Murs ». Pour chaque pixel de la grille, l'altitude minimale et l'altitude maximale des points compris dans ce pixel sont déterminées. Sachant que l'extension d'un mur est verticale, la différence entre ces deux valeurs doit être importante. Dès lors, les pixels ne présentant pas une différence d'altitude suffisante ne font pas partie d'un mur. Les points contenus dans ces pixels sont donc exclus de cette classe.

Au final, les points rejetés lors des différentes étapes sont regroupés ensemble pour constituer le reste du nuage et participer à la suite des traitements. Les autres points sont attribués à la classe « Murs ».

4.1.4.2. Portes

L'extraction des portes se fait après celle des murs, sur les points restants. Cela permet de travailler sur un nuage plus petit et de ne pas modifier la classe des points attribués aux murs. Une cible ayant été apposée sur chaque porte, le plan de la porte peut être extrait en déterminant les paramètres du plan défini sur base du voisinage du centre de la cible (4). Pour prendre en compte l'épaisseur de la porte, les points attribués à cette classe sont ceux situés en-dessous d'une certaine distance de ce plan (5). A nouveau, un plan n'étant pas limité en extension, des points extérieurs à la porte se verraient également attribuer cette classe. Afin d'éviter ce problème, une condition de distance à la cible est ajoutée à la condition d'appartenance au plan.

4.1.4.3. Fenêtres

La segmentation des fenêtres se fait à partir des deux cibles disposées sur la même façade. En combinant les voisins de ces deux cibles, le meilleur plan passant par ces points est déterminé avec la fonction (4). Ce plan permet ainsi d'extraire l'intégralité des fenêtres présentes sur la façade, à condition qu'elles soient bien alignées dans un même plan. A nouveau, les points situés à une distance du plan inférieure à un seuil fixé (5) sont extraits pour tenir compte de l'épaisseur des châssis. Pour éviter de considérer les points du plan en-dehors de la façade, des conditions en X, Y et Z sont également appliquées. Les coordonnées minimales et maximales des deux cibles sont calculées et une marge de sécurité leur est appliquée. Dès lors, un point est attribué à la classe « Fenêtres » s'il remplit les quatre conditions suivantes :

1) $d(\text{plan}, \text{pt}) < \text{seuil}$
2) $X_{min} - \text{marge} < X_{pt} < X_{max} + \text{marge}$
3) $Y_{min} - \text{marge} < Y_{pt} < Y_{max} + \text{marge}$
4) $Z_{min} - \text{marge} < Z_{pt} < Z_{max} + \text{marge}$

4.1.4.4. Gouttières

Les gouttières, s'il y en a, sont ensuite extraites une à une du reste du nuage en utilisant le principe de croissance de région (*Region Growing*). Pour chaque gouttière, la graine utilisée

comme point de départ de la croissance de région correspond au point du nuage le plus proche du centre de la cible. A partir de ce point, les points voisins sont ajoutés à la région s'ils respectent deux conditions : leur normale doit être proche de l'horizontale et l'angle entre celle-ci et la normale du point précédent doit être inférieur à une valeur fixée. Cela permet notamment de distinguer le sol de la gouttière. Le rayon de voisinage est également important puisque c'est lui qui détermine la distance maximale entre deux points d'une même région. Il doit donc être suffisamment petit pour ne pas prendre en compte des points extérieurs à la gouttière.

Néanmoins, le calcul des normales (3) étant chronophage et gourmand en mémoire, celui-ci n'a pas été effectué sur l'ensemble du nuage de points. Afin d'accélérer les traitements, un sous-nuage a été préalablement extrait en conservant uniquement les points situés sous une certaine distance (horizontale) des cibles disposées sur les gouttières. Les normales des points ne sont donc calculées que pour ce sous-nuage. C'est également à partir de ce sous-nuage que les gouttières sont extraites.

A l'issue de ces traitements, chaque point du nuage se voit attribuer un numéro correspondant à la classe à laquelle il appartient. Dans le cadre de ce travail, les cinq classes suivantes ont été utilisées :

1 = murs

2 = portes

3 = fenêtres

4 = gouttières

0 = autres objets et bruit de mesure

Les résultats sont ensuite enregistrés dans un fichier reprenant les coordonnées X, Y et Z de chaque point ainsi que la classe associée à ce point.

4.2. Méthode par levé topographique

La seconde méthode développée dans ce travail est plus traditionnelle et proche des missions confiées au géomètre-expert puisqu'elle est basée sur la réalisation d'un levé topographique avec géocodification. Son objectif est de faciliter le traitement d'un nuage de points en levant des points spécifiques auxquels des codes sont associés. Dans les parties concernant l'acquisition et le prétraitement des données, il faut inclure les traitements appliqués au nuage de points qui ont été détaillés dans la méthode précédente. En effet, le même nuage a été utilisé pour les deux méthodes afin de pouvoir comparer les résultats.

4.2.1. Géocodification

A nouveau, la première étape est la création de la géocodification. Celle-ci doit être établie en fonction des éléments présents dans la zone et de l'objectif final de la mission. Comme expliqué pour la première méthode, nous nous sommes concentrés sur le bâti. Les éléments d'intérêt à distinguer du reste sont donc les murs, les portes et les fenêtres ainsi que les gouttières éventuelles.

L'établissement de la géocodification a été fait après une certaine réflexion sur le processus de segmentation/classification. Il faut déterminer au préalable comment les données vont être traitées pour développer une méthodologie cohérente et adaptée à la suite du travail. Chaque objet se voit donc attribuer une méthode de lever et une codification spécifique sur base des concepts présentés dans la partie 2.1.1.

L'emprise d'un bâtiment étant une entité polygonale, elle peut être vue comme une polyligne fermée. Une liaison semble donc appropriée pour ce type d'objets. Pour rappel, une liaison est composée d'une séquence et d'un modificateur. Plusieurs séquences (codes) sont attribuées afin de pouvoir lever plusieurs bâtiments en même temps si nécessaire. Un modificateur de début est utilisé afin de distinguer plusieurs bâtiments qui seraient levés avec la même séquence. Un modificateur différent est assigné aux autres points délimitant l'emprise pour assurer une continuité dans le levé. De cette manière, la géocodification est adaptée à tous types de bâtiments, quels que soient leur forme ou leur nombre. La figure 8 illustre ces concepts. La station rouge représente le point de départ du levé permettant de mesurer partiellement le premier bâtiment avec la séquence 100. Celui-ci n'étant néanmoins pas levé complètement, le deuxième bâtiment doit commencer avec une autre séquence (101). En passant à la seconde station, les deux bâtiments entamés peuvent être clôturés. Le levé du

troisième bâtiment peut alors commencer avec la même séquence que le premier (100). Dès lors, le rôle du modificateur est indispensable pour distinguer les points appartenant à chacun des bâtiments. Ainsi, le modificateur du premier point d'un bâtiment (.2 dans l'exemple) doit être différent de celui appliqué aux autres points (.4).

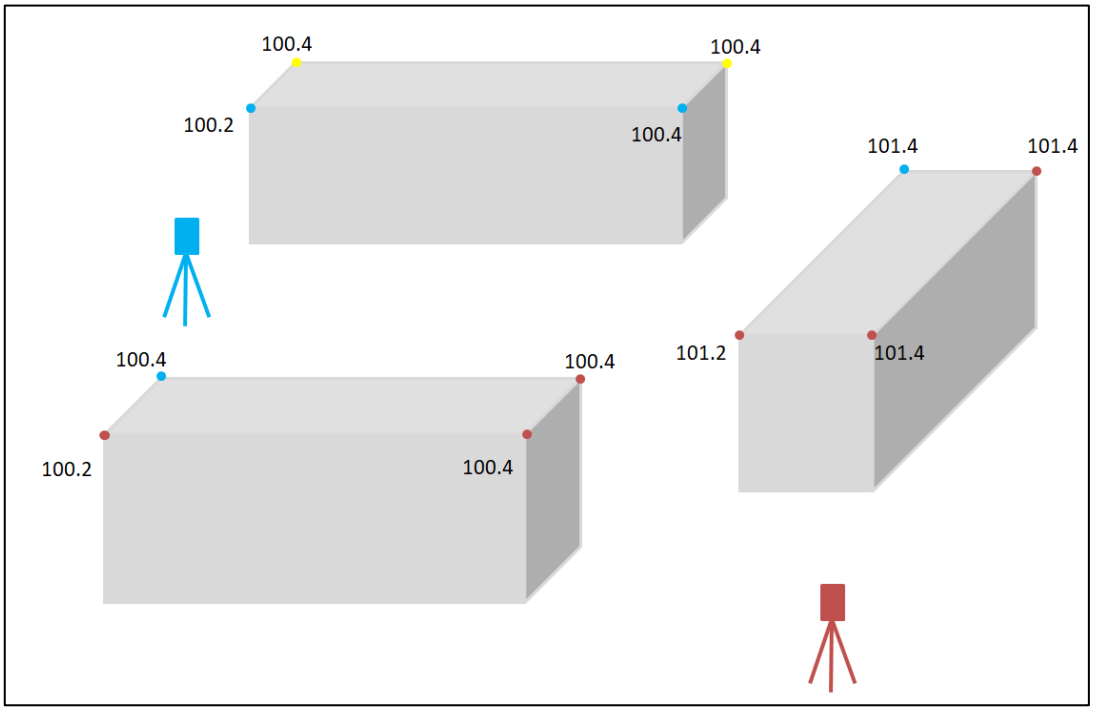


Figure 8 : Exemple de géocodification à appliquer lors du levé de bâtiments

La géocodification des portes et des fenêtres est similaire. Elles sont dès lors détaillées sous le terme d'ouvertures. Le code appliqué au final sera néanmoins différent afin de les distinguer en deux classes. Trois cas différents peuvent se présenter. Si les ouvertures sont rectangulaires, elles peuvent être simplement délimitées par deux points opposés (le coin en haut à gauche et le coin en bas à droite par exemple). Le même code est alors attribué pour les deux points de chaque ouverture, c'est le premier cas illustré à la figure 9. Le second cas se présente lorsqu'un des coins n'est pas visible. En effet, comme les ouvertures sont généralement en retrait par rapport à la façade, la position de l'appareil de mesure influence directement la visibilité des extrémités de l'ouverture. Pour pallier à ce problème, un code différent est prévu et, avec lui, la mesure d'un troisième point. La méthode consiste donc à mesurer deux points de l'ouverture au niveau de la façade et un troisième, affecté d'un modificateur particulier, au niveau de l'extrémité en retrait visible. Ainsi, le vecteur de déplacement entre l'extrémité en retrait et celle en façade pour le même coin permettra d'obtenir l'extrémité manquante en post-traitement (en rouge sur la figure ci-dessous). Le dernier cas est valable pour les ouvertures de formes complexes dont le contour serait mesuré

à l'aide d'une polyligne fermée. La géocodification pour ce type d'ouvertures est donc semblable à celle présentée précédemment pour les bâtiments, avec l'utilisation d'une séquence et d'un modificateur.

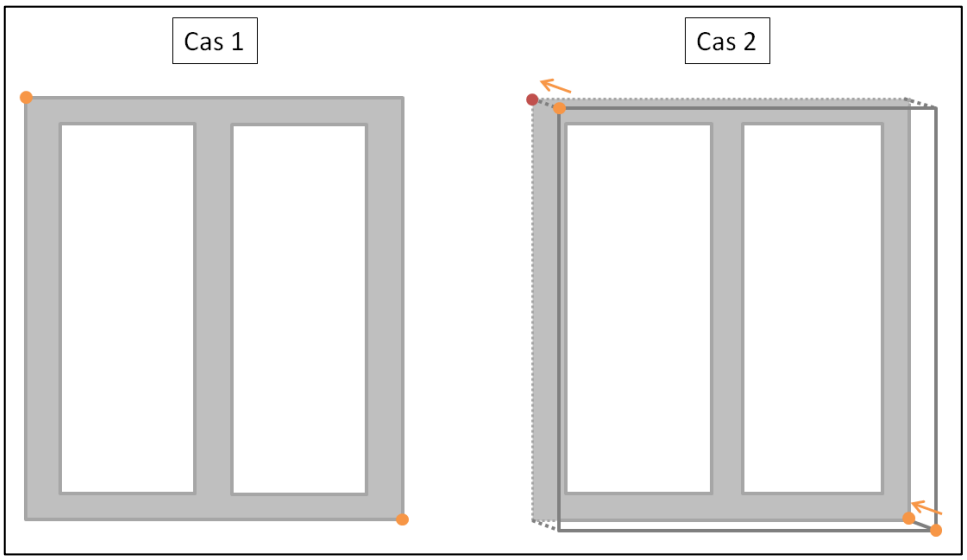


Figure 9 : Points à lever pour une ouverture rectangulaire : deux cas possibles en fonction de la position de l'appareil

En ce qui concerne les gouttières, elles peuvent globalement être représentées par des cylindres. Trois paramètres permettent de décrire un cylindre : le centre, le rayon et la hauteur/longueur. Un code spécifique est attribué à chacun de ces paramètres. En cas de gouttière plus complexe présentant des coudes, il suffirait de mesurer des points au niveau des changements de direction en suivant une liaison.

La même géocodification conviendrait pour les tuyaux, également cylindriques. Pour d'autres types d'objets, des codes supplémentaires devraient être déterminés et adaptés en fonction de la géométrie de l'objet et de la méthode de segmentation choisie. La géocodification utilisée dans ce travail est présentée dans la partie consacrée aux résultats (cf. 6.2.1.).

4.2.2. Acquisition

Pour cette seconde méthode, l'acquisition des données sur le terrain se divise en deux parties puisque deux types de données sont nécessaires. Tout d'abord, il faut collecter un nuage de points de la zone tel qu'expliqué au point 4.1.2. Ensuite, sur base de la géocodification établie précédemment, il faut procéder au levé des points utiles pour la segmentation et la classification des éléments d'intérêt. C'est cette partie de l'acquisition qui est détaillée ci-dessous. Néanmoins, des points communs fixes sont nécessaires entre l'acquisition au

laserscan et celle à la station totale pour faciliter la suite des traitements et travailler dans un système unique.

Afin de lever les différents points dans un même système de coordonnées, une polygonale est établie de manière à couvrir toute la zone. Celle-ci est matérialisée à l'aide de clous ce qui permet, en cas de problèmes ou de mesures supplémentaires nécessaires, de retourner sur le terrain et de se repositionner dans le même système. La précision des points levés est non-seulement dépendante des caractéristiques de l'appareil utilisé mais également de la précision des points de la polygonale, à partir desquels les mesures sont faites. Dès lors, afin d'obtenir des coordonnées de la meilleure qualité possible, un maximum d'inter-visées entre les points matérialisés sont recommandées.

Le levé des éléments à segmenter est ensuite effectué en appliquant la géocodification créée et adaptée pour la zone et les éléments qu'elle comprend.

Les bâtiments sont levés en mesurant les points aux extrémités supérieures des murs qui délimitent leur emprise (coins hauts). Cela permet, si le bâtiment a un toit et s'il est mesurable au scanner laser terrestre, de distinguer directement les murs du toit. Un autre point est mesuré au niveau d'un coin bas du bâtiment. Ce dernier n'est pas impératif si le bâtiment est au niveau du sol, ce qui est généralement le cas, mais cette méthodologie permet de prendre en compte tout type de bâtiment, y compris des bâtiments sur pilotis.

Les portes et les fenêtres sont levées en deux, trois et parfois même quatre ou plus de points selon la configuration rencontrée. Si l'appareil de mesure est situé en face de l'ouverture à lever, toutes les extrémités sont visibles et il suffit de mesurer deux points opposés, c'est la configuration la plus simple et la plus rapide. Par contre, si l'appareil est décalé par rapport à l'ouverture (visées obliques), trois points sont mesurés : deux points délimitant l'ouverture au niveau du mur et une extrémité visible dans le renfoncement afin de connaître la profondeur. Pour des ouvertures plus complexes, plus de points sont nécessaires. Par exemple, quatre points sont levés lorsque l'appui de fenêtre est incliné ce qui implique que l'ouverture au niveau du mur n'est pas identique à la taille de la fenêtre. Il faut également mesurer plus de points en cas de formes géométriques plus complexes. L'ordre n'a pas d'importance pour les fenêtres mais bien pour les portes. En effet, afin de connaître le côté d'ouverture d'une porte, c'est d'abord le point du côté de la charnière qui est mesuré. Ces trois types d'objets (murs, portes et fenêtres) sont levés au laser puisqu'il est difficile de positionner un prisme au niveau des points à mesurer. En outre, cela permet une prise de mesures plus rapide.

Pour les gouttières, un prisme est utilisé afin de pouvoir mesurer le centre du tuyau ce qui n'est pas possible au laser. Le bord du tuyau est également mesuré pour pouvoir déterminer son rayon à posteriori. Ces deux points sont levés en bas de la gouttière. Enfin, un troisième point est levé afin de connaître la hauteur de la conduite.

L'acquisition est une étape importante dans le processus ; elle doit être complète puisque seuls les éléments levés seront traités par la suite. Cela permet de traiter uniquement les éléments d'intérêts et non la totalité du nuage de points. Néanmoins, cela implique que, en cas d'oubli ou d'obstacles empêchant certaines mesures, les objets manquants ne participeraient pas à la suite des traitements.

4.2.3. Prétraitements

Les mesures effectuées lors du levé doivent ensuite être traitées avant de pouvoir les utiliser pour le processus de segmentation/classification. Les prétraitements appliqués au nuage de points, détaillés dans la partie 4.1.3, sont également à considérer.

Le premier prétraitement consiste à corriger les éventuelles erreurs commises lors de l'acquisition. Celles-ci peuvent être de différentes natures telles que l'application d'un mauvais code, l'enregistrement d'une hauteur d'instrument (station ou prisme) erronée ou encore un ordre de points à lever non respecté. Ces corrections doivent être faites manuellement.

Après avoir rectifié ces erreurs, les coordonnées des points sont calculées. Les coordonnées de la polygonale sont obtenues par application du principe des moindres carrés afin de minimiser les résidus d'observations. Les inter-visées entre points effectuées lors de l'acquisition permettent d'effectuer l'ajustement grâce aux mesures redondantes. Celui-ci renvoie non-seulement les coordonnées ajustées mais également les précisions associées à ces coordonnées. Ces coordonnées peuvent soit être exprimées dans un système de référence local, soit dans un système général tel que le Lambert 2008. Ce choix dépend de l'objectif du travail, de la zone étudiée et de la présence (ou non) de points connus dans un système particulier et matérialisés sur le terrain. Quel que soit le système utilisé, il doit être identique pour les points topographiques levés et pour le nuage de points. A cet effet, l'acquisition des deux jeux de données comprend au moins deux points communs fixes (clous), inclus dans la polygonale. Les points de la polygonale étant désormais connus, les coordonnées X, Y et Z

de tous les points levés sur le terrain sont calculées sur base des mesures d'angles et de distances.

L'issue de ces prétraitements est un fichier structuré de la manière suivante :

ID ; X ; Y ; Z ; code

A chaque ligne correspond un point levé différent. Ce fichier, combiné au nuage de points traité et positionné dans le même système de référence, servira de base au processus de segmentation/classification.

4.2.4. Segmentation et classification

Comme pour la méthode précédente, la géocodification permet de connaître directement les classes attendues en résultat. La segmentation peut donc être adaptée à chaque type d'objet (classe) en tenant compte de ses caractéristiques géométriques et de la position des points levés. Trois fonctions de segmentation principales ont été implémentées pour correspondre aux objets présents : une *bounding box 3D*, une extraction rectangulaire et une extraction cylindrique. D'autres fonctions pourraient être créées pour correspondre à d'autres éléments à segmenter. Le nuage de points a été segmenté et classifié classe par classe. Dès lors, avant le processus de segmentation en tant que tel, le fichier contenant les coordonnées et les codes des points levés a été trié selon ces derniers et divisé afin de séparer les différents types d'objets. De cette manière, la fonction de segmentation adaptée à chaque classe peut être appliquée sur l'ensemble des objets lui appartenant. Pour faciliter les traitements, une colonne a également été ajoutée afin de séparer le code en séquence et modificateur. Les données utilisées pour la segmentation sont donc triées en fonction de leur code et organisées en six colonnes :

ID | X | Y | Z | séquence | modificateur

4.2.4.1. Bounding Box 3D

Cette méthode de segmentation a été développée pour les objets volumiques 3D dont l'emprise au sol est rectiligne et dont la hauteur est connue et mesurable tels que des bâtiments entiers ou des cabines électriques. Cette fonction nécessite les coordonnées des points délimitant l'emprise de l'objet ainsi que son altitude minimale et maximale. Une marge de sécurité est appliquée sur l'emprise et sur les altitudes pour ne pas supprimer des

points faisant partie de l'objet. Dès lors, les points compris dans l'emprise 2D (conditions sur X et Y) et dont la coordonnée Z se trouve entre la valeur minimale et maximale sont retenus et extraits du nuage de points. Cette fonction peut être réappliquée sur les points extraits afin d'éliminer les points intérieurs inutiles. En effet, pour les bâtiments par exemple, il y a présence de bruit lorsque le signal du laserscan traverse une fenêtre et est renvoyé par un objet intérieur. Pour éliminer ces points, la fonction est réutilisée en décalant l'emprise d'une distance proche de l'épaisseur des murs et en supprimant cette fois les points situés à l'intérieur du volume. Ainsi, l'objet segmenté correspond au bâtiment avec ses murs, ses portes et fenêtres, mais sans le bruit intérieur.

4.2.4.2. Extraction rectangulaire

Certains éléments peuvent être segmentés sur base d'une délimitation rectangulaire, c'est par exemple le cas des portes et des fenêtres. A partir de deux points situés aux extrémités de l'objet (un en haut à gauche et un en bas à droite par exemple), celui-ci peut être extrait. Pour chaque coordonnée X, Y et Z, les valeurs minimales et maximales entre ces deux points sont déterminées. Ainsi, un point appartenant à l'objet (pt) doit remplir les trois conditions suivantes:

1) $X_{min} < X_{pt} < X_{max}$
2) $Y_{min} < Y_{pt} < Y_{max}$
3) $Z_{min} < Z_{pt} < Z_{max}$

Pour assurer la prise en compte de tous les points, une petite marge de sécurité est appliquée sur les coordonnées minimales et maximales. Néanmoins, ces conditions ne suffisent pas puisque, en fonction de la position de l'objet par rapport au système de référence, une zone plus ou moins grande est prise en compte. La figure 10 met en évidence cette problématique avec deux segments de droites orientés différemment par rapport aux axes X et Y ce qui engendre un rectangle capable différent (en gris sur la figure). Des points extérieurs à l'objet pourraient dès lors être inclus. Pour éviter ce problème, une quatrième condition sur la distance entre le point analysé et le segment de droite liant les deux points extrêmes (1-2) doit être remplie :

4) $d(1\text{-}2, pt) < seuil$

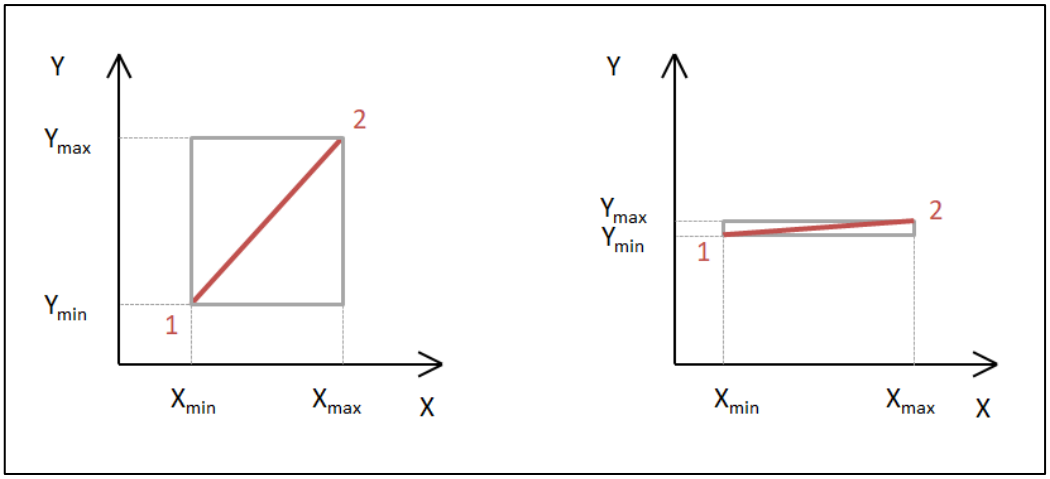


Figure 10 : Illustration de l'influence de l'orientation d'un segment sur les dimensions du cadre capable

4.2.4.3. Extraction cylindrique

La dernière fonction de segmentation implémentée dans ce travail est consacrée à l'extraction d'objets cylindriques tels que des gouttières, des tuyaux ou encore des poteaux, à partir de trois points mesurés. Tout d'abord, le rayon de l'objet peut être calculé à partir des coordonnées du centre de l'objet et du point mesuré sur la circonférence. Une marge de sécurité est ajoutée au rayon obtenu pour éviter le rejet des points situés à l'extrémité de l'objet. Le troisième point mesuré permet de connaître la hauteur maximale des points appartenant à l'objet alors que la hauteur minimale correspond à la coordonnée Z du centre. Ainsi, la segmentation est faite en combinant les deux conditions suivantes :

1) d(centre, pt) < rayon + marge
2) $Z_{min} < Z_{pt} < Z_{max}$

Cela signifie que les points sont attribués à l'objet s'ils sont situés à une distance de son centre inférieure à la valeur du rayon augmenté d'une marge et si leur coordonnée Z se trouve dans l'intervalle défini par les points levés.

Chaque point du nuage se voit ainsi attribuer un numéro entre 0 et 4 correspondant à la classe à laquelle il appartient. Les classes sont les mêmes que celles clôturant la méthode précédente. Les résultats sont ensuite enregistrés dans un fichier reprenant les coordonnées X, Y et Z de chaque point ainsi que la classe associée à ce point.

4.3. Comparaison et validation des résultats

Le processus ne s'arrête pas là. Les résultats des deux méthodes implémentées doivent désormais être validés et comparés. La validation s'effectue à partir d'un nuage de référence, également appelé « vérité terrain », qui permettra d'analyser la qualité et la justesse des résultats. Cette vérité terrain est créée en segmentant et classifiant manuellement le nuage de points sur base des cinq classes citées précédemment. Le résultat issu de l'algorithme de segmentation est dès lors comparé à cette vérité terrain. Plusieurs métriques existent pour quantifier la qualité des résultats, elles peuvent être calculées à partir des informations contenues dans une matrice de confusion.

Une matrice de confusion est une matrice carrée de dimension égale au nombre de classes. Elle permet de comparer le nuage de référence et le nuage classifié en reprenant le nombre de points bien et mal classés pour les différentes classes. Dès lors, il faut s'assurer de comparer les classes correspondant aux mêmes points de chaque nuage. Les deux nuages sont donc préalablement triés sur base de leurs coordonnées afin de comparer les points homologues. Un exemple de matrice de confusion est repris ci-dessous. Plusieurs termes peuvent être extraits de cette matrice (Lesecque, 2019 ; Poux & Billen, 2019) :

- Vrai Positif (VP) : observation attribuée à la classe à laquelle elle appartient vraiment (en vert sur la figure).
- Faux Positif (FP) : observation attribuée à la classe étudiée (classe 1 dans l'exemple) alors qu'elle appartient en réalité à une autre (en bleu).
- Faux Négatif (FN) : observation attribuée à une classe différente alors qu'elle appartient en réalité à la classe étudiée (en rouge).
- Vrai Négatif (VN) : observation appartenant à une classe différente de celle étudiée et attribuée effectivement à une classe différente.

		Résultat de la classification				
		Classe 0	Classe 1	Classe 2	Classe 3	Classe 4
Vérité terrain	Classe 0					
	Classe 1					
	Classe 2					
	Classe 3					
	Classe 4					

Tableau 1 : Exemple d'une matrice de confusion vide

A partir de ces termes, plusieurs métriques couramment utilisées dans la littérature pour analyser la qualité d'un algorithme de classification peuvent être calculées (Vo *et al.*, 2015 ; Nurunnabi *et al*., 2016 ; Jakovljevic *et al.*, 2019 ; Poux & Billen, 2019) :

$$oAcc = \frac{\sum_{i=1}^{c} x_{ii}}{n} \quad (4)$$

$$Precision = \frac{VP}{VP+FP} \quad (5)$$

$$Recall = \frac{VP}{VP+FN} \quad (6)$$

$$F_{1-score} = 2 * \frac{precision * recall}{precision + recall} \quad (7)$$

L'exactitude globale (*oAcc* pour *Overall Accuracy*) est une mesure générale de la qualité de la classification puisqu'elle représente la proportion de points bien classés dans le résultat. Il s'agit de la somme des éléments diagonaux ($\sum_{i=1}^{c} x_{ii}$) divisé par la totalité des points du nuage (n). Puisqu'il s'agit d'une mesure globale, elle est fortement affectée par la taille des classes. Si celles-ci sont inégales, ce qui est généralement le cas, le résultat sera biaisé par la classe dominante, regroupant le plus de points. Pour pallier à ce problème, les trois autres métriques sont calculées pour chaque classe séparément. La précision représente la part de points correctement classés parmi tous les points attribués à une classe particulière. Le rappel (*recall*) correspond au pourcentage de points d'une classe de vérité terrain qui sont correctement classés par l'algorithme. La qualité de la classification augmente lorsque ces métriques se rapprochent d'une valeur unitaire. Le F1-score combine habilement ces deux notions pour fournir une mesure unique de l'efficacité de la classification.

L'évaluation des algorithmes ne s'arrête toutefois pas là. Outre la qualité des résultats, ce travail s'intéresse à la rapidité des traitements. En effet, l'objectif de ce mémoire est de développer une (ou plutôt deux) méthode(s) permettant de segmenter et classifier un nuage de points de manière plus rapide qu'une segmentation manuelle sans pour autant altérer significativement la qualité des résultats. Par conséquent, les temps totaux pour chacune des deux méthodes doivent être comparés avec le temps requis pour réaliser la segmentation manuelle. Ceux-ci doivent évidemment tenir compte du temps nécessaire pour l'acquisition, pour les prétraitements et pour la segmentation.

5. ENVIRONNEMENT DE TRAVAIL

Cette partie est consacrée à la description du contexte au sein duquel la méthodologie a été développée. Cet environnement de travail comprend la zone étudiée ainsi que le matériel et les logiciels employés. Les données utilisées pour la segmentation seront présentées dans la partie « Résultats » puisqu'elles ont été récoltées dans le cadre même de ce travail.

La zone étudiée dans ce travail se situe sur le campus du Sart-Tilman de l'Université de Liège, au niveau de la Traverse des Architectes. Elle a été choisie puisqu'il s'agit d'une zone facilement accessible, connue et composée de plusieurs petits bâtiments aisément mesurables au laserscan terrestre. De plus, des mesures y ont déjà été réalisées auparavant ce qui permettait de disposer de données d'entrainement avant de procéder à l'acquisition définitive. Des points de coordonnées Lambert 2008 connues étaient également matérialisés sur le terrain, facilitant l'établissement de la polygonale nécessaire au développement de la deuxième méthode. Initialement, l'idée était d'étudier l'intégralité des bâtiments encadrés en rouge sur la figure ci-dessous. Cependant, par manque de temps, le nuage de points collecté sera limité à un des bâtiments et aux façades des autres bâtiments visibles depuis ce dernier (en jaune). Ceci sera détaillé dans le chapitre suivant (cf. 6. Résultats).

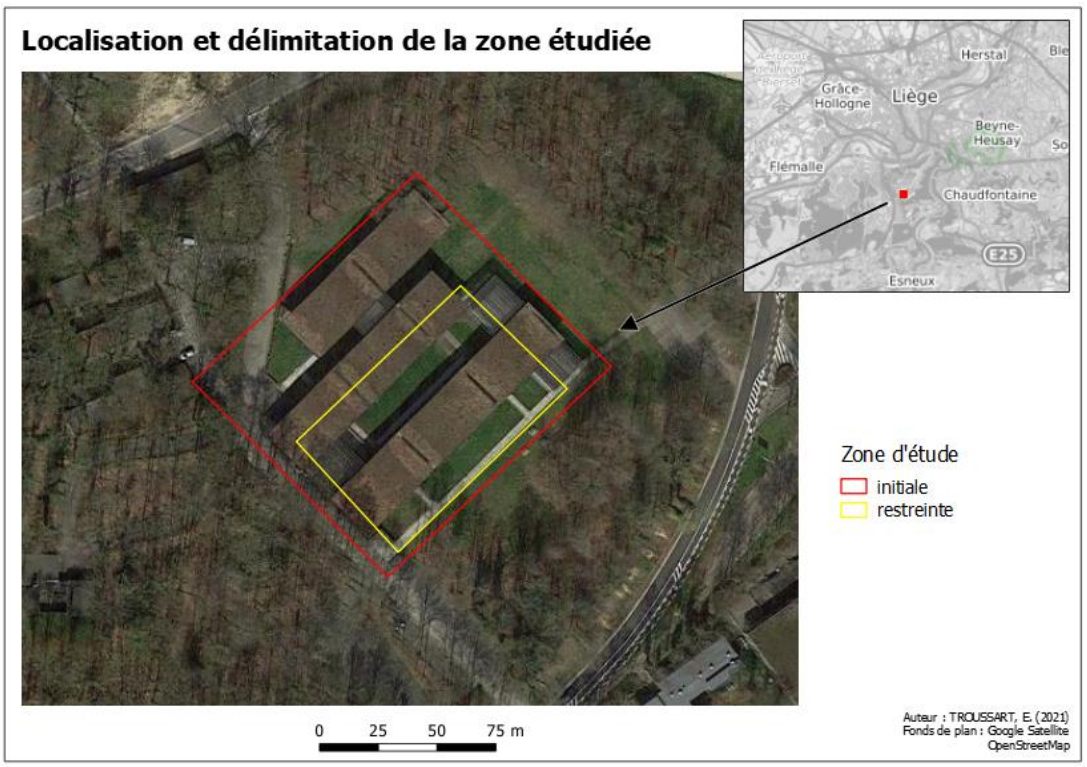


Figure 11 : Localisation et délimitation de la zone

Le matériel employé pour procéder à l'acquisition est celui mis à disposition par l'Université de Liège. Pour le levé topographique, il s'agit d'une station totale Leica 1200 accompagnée d'un trépied et d'un mini-prisme. Celle-ci n'étant pas automatique, un accompagnateur devait être présent pour effectuer les mises en station et le levé des gouttières au prisme. Néanmoins, comme la plupart des géomètres travaillent avec des stations automatiques, il n'est pas indispensable d'être deux pour réaliser les mesures. En ce qui concerne l'acquisition du nuage de points, l'appareil utilisé est un scanner laser terrestre Leica P30, accompagné de ses cibles et d'un trépied. Des cibles supplémentaires ont été créées et imprimées sur papier pour appliquer la géocodification de la méthode des cibles.

L'ordinateur utilisé pour effectuer les différents traitements dispose de 16 gigas de RAM, d'un processeur Intel Core i7 et d'une carte graphique NVIDIA GeForce MX230. Plusieurs logiciels ont été nécessaires pour la réalisation des différentes étapes de ce travail.

Tout d'abord, deux logiciels propriétaires ont été utilisés pour le prétraitement des données : *Cyclone* et *Covadis*. Le premier est destiné au traitement et à la gestion des nuages de points. Il a été utilisé pour importer et assembler les différents scans sur base des cibles, extraire les coordonnées de ces dernières et nettoyer grossièrement le nuage afin de limiter son volume. Le second a servi pour calculer les coordonnées des points du levé topographique ainsi que la précision associée aux points de la polygonale.

Des logiciels *Open Source* ont également été utilisés. Le logiciel *CloudCompare* (CloudCompare (2.10.2), 2021) a été employé pour la visualisation et la manipulation du nuage de points tout au long du travail. C'est avec ce dernier que la segmentation manuelle a été réalisée. Enfin, *Anaconda Navigator* (Anaconda Navigator (1.10.0), 2020) a été utilisé pour la gestion des librairies avec *Spyder 3.3.4* comme environnement de développement. Les scripts, repris aux annexes 10.1 à 10.3, ont été réalisés en Python (version 3.7). Plusieurs librairies ont été installées pour le développement des algorithmes :

- NumPy (Harris *et al.*, 2020) : manipulation et opérations diverses avec des tableaux multidimensionnels
- Laspy (Brown & Butler, 2014) : lecture et écriture des fichiers .las
- SciPy (Virtanen *et al.*, 2020) : calcul de distances
- Matplotlib (Hunter, 2017) : histogramme et opérations topologiques

- Scikit-learn (Pedregosa *et al.*, 2011) : calcul des métriques pour la validation des résultats
- Math : fonctions mathématiques
- Time : calcul des temps de traitements
- Random : génération de nombres aléatoires (pour appliquer RANSAC).

6. RÉSULTATS

Les résultats obtenus tout au long de ce travail sont présentés dans ce chapitre. Pour suivre l'ordre de la méthodologie, ils sont détaillés par étape, d'abord pour la méthode des cibles et ensuite pour la méthode par levé topographique. Cette partie inclut donc les géocodifications créées, les données issues de l'acquisition et traitées afin d'être utilisées pour la segmentation ainsi que les résultats des algorithmes de segmentation/classification. A nouveau, le nuage de points sera uniquement présenté dans la première méthode puisque c'est le même qui a servi pour la seconde. La validation et la comparaison des résultats concluront ce chapitre.

6.1. Méthode des cibles

6.1.1. Géocodification

La géocodification utilisée pour cette méthode est relativement simple puisqu'un code unique est attribué à chaque cible. Ce code est déterminé en fonction de l'objet sur lequel la cible sera disposée. Cinq codes ont été employés pour distinguer les quatre classes d'intérêt et les cibles servant uniquement à l'assemblage des nuages de points. Le tableau ci-dessous présente les codes associés à chaque objet avec une petite description concernant la position et le nombre de cibles à placer. Le détail complet se trouve dans la partie méthodologie (cf. 4.1.1.).

Objets	Description	Code
Bâtiment/murs	1 cible en bas à gauche de chaque mur	1
Portes	1 cible par porte	2
Fenêtres	1 cible sur chaque fenêtre située aux extrémités de la façade	3
Gouttières	1 petite cible par gouttière	4
Cibles supplémentaires	Cibles d'assemblage	99

Tableau 2 : Géocodification utilisée pour la méthode des cibles

Les quatre codes attribués aux objets d'intérêt correspondent aux valeurs qui seront utilisées pour la classification. Il faut ajouter à ces codes la classe 0 qui contiendra le reste des points.

6.1.2. Acquisition et prétraitements

L'acquisition au laserscan terrestre n'a pas été effectuée sur la totalité de la zone initialement prévue. En effet, afin de limiter le temps consacré à l'acquisition par rapport à celui nécessaire pour les traitements, la zone a été restreinte à un bâtiment et aux façades avoisinantes des trois autres bâtiments. Malheureusement, ce choix ayant été fait tardivement (à l'issue de la première journée d'acquisition du nuage de points), le placement des cibles en a été impacté. Certains éléments, devant au départ être mesurés depuis d'autres points de vue lors d'une deuxième journée d'acquisition, ne se sont pas vus attribuer de cibles. D'un autre côté, des cibles placées en prévision d'un futur scan ne sont pas détectables sur le nuage récolté. C'est par exemple le cas de deux cibles positionnées sur des gouttières situées trop loin de l'appareil. Pour limiter l'impact de ce changement sur les résultats, le nuage total sera divisé en deux nuages distincts qui seront analysés séparément. Le premier sous-nuage reprendra le bâtiment mesuré entièrement. Le deuxième sera consacré au reste des données ce qui permettra d'analyser l'algorithme sur des bâtiments incomplets.

L'acquisition en tant que telle comprend deux parties : la disposition des cibles et la prise de mesures au laserscan. Le positionnement des cibles a été fait au fur et à mesure afin d'assurer un minimum de cinq cibles communes entre deux scans successifs. Tout d'abord, les cibles accompagnées d'un code ont été placées sur les éléments particuliers (murs, portes, fenêtres et gouttières). Ensuite, les cibles supplémentaires ont été disposées de manière à assurer une bonne répartition spatiale et un recouvrement suffisant. Lorsque des points de la polygonale étaient accessibles, une cible fixée sur une canne y était positionnée. Au total, 54 cibles ont été placées dans la zone restreinte : 30 pour la segmentation, 20 pour la consolidation et 4 pour le géoréférencement. Sept scans ont été réalisés pour collecter l'intégralité des données et limiter les occlusions. La figure 12 reprend la position de ces sept scans et des quatre points/cibles utilisés pour le géoréférencement.

Il a fallu environ 4h30 de travail pour disposer les cibles et procéder à l'acquisition des données au laserscan sur la zone restreinte.

Les données ont ensuite été traitées afin de pouvoir procéder à la segmentation. Après avoir importé les différents scans dans le logiciel *Cyclone* et après avoir vérifié la position des cibles, l'assemblage a été effectué pour obtenir un seul et unique nuage de points. Afin de procéder au géoréférencement de ce nuage, les coordonnées Lambert 2008 des quatre cibles placées sur les points 1, 2, 11 et 15 de la polygonale (cf. 6.2.2.) ont été encodées. Connaissant

les coordonnées de ces quatre points dans les deux systèmes, les paramètres de la transformation à appliquer ont été déterminés par moindres carrés. A l'issue de cette résolution, les résidus d'observations sur les quatre points sont de l'ordre de 5 mm. Ceux-ci découlent d'une combinaison de plusieurs petites erreurs. En effet, l'imprécision sur les coordonnées des points de la polygonale entre en ligne de compte tout comme celle provenant du positionnement des cibles sur ces points. Il faut également y ajouter les erreurs résultant de la consolidation des scans.

Le résultat de cette étape est un nuage complet, rattaché au système de référence désiré, mais contenant de nombreux points inutiles.

Figure 12 : Positions des stations de scans (en bleu) et des points de référence (en rouge)

Afin de diminuer le volume des données, un nettoyage rapide du nuage a été réalisé. La majorité de la végétation située aux abords de la zone a été supprimée tout comme une partie du sol situé à distance des bâtiments. Le bruit provenant de la réflexion du laser dans les fenêtres a également été retiré. Le nuage a ensuite été échantillonné de manière à conserver un point tous les 4 mm. Après ces traitements, le nuage de points complet a été exporté au format .las. Néanmoins, comme *Cyclone* est un logiciel propriétaire, ces traitements ont été effectués sur un ordinateur différent de celui utilisé pour la segmentation. Il s'est avéré par la suite que le nuage exporté était toujours trop volumineux pour être manipulé sur mon

ordinateur. D'autres points ont donc été retirés et un deuxième échantillonnage, aléatoire cette fois, a été fait pour l'alléger. Ainsi, d'un nuage de 433 110 060 points nous sommes passés à un nuage de 125 000 000 points tel qu'illustré ci-dessous.

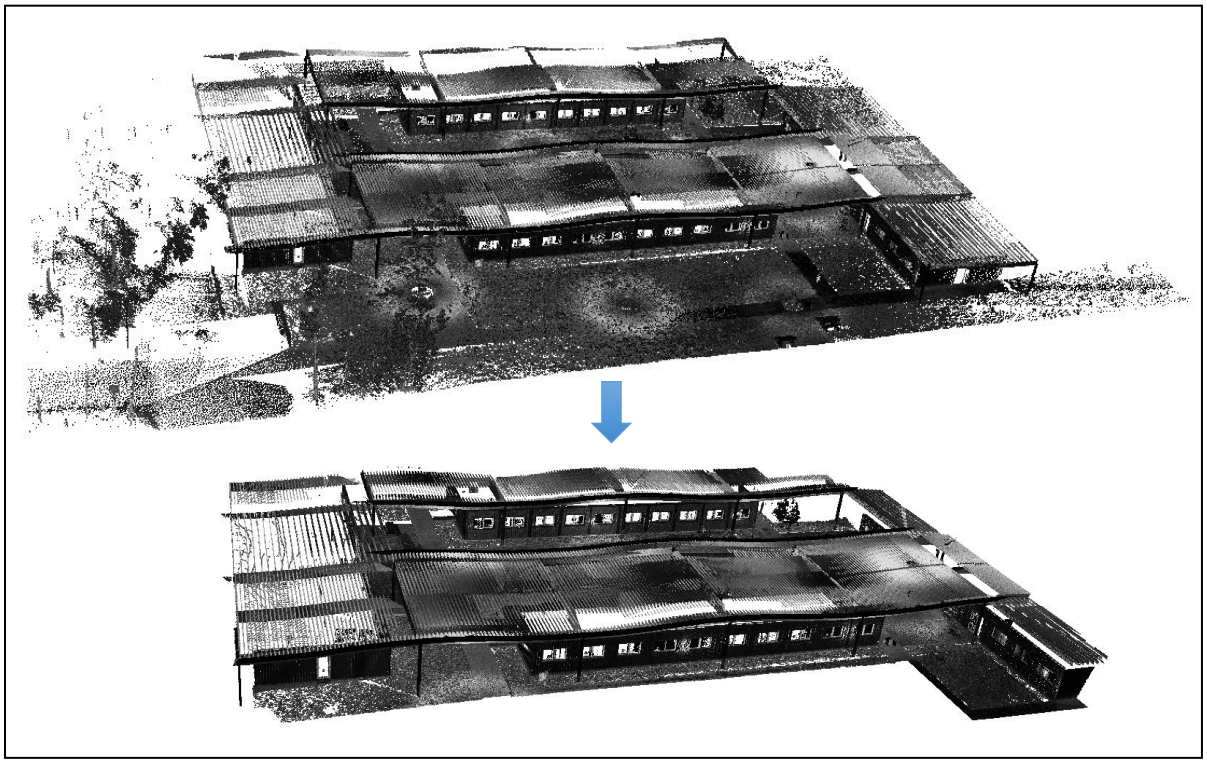


Figure 13 : Réduction de la taille du nuage de points (433 110 060 → 125 000 000 points)

Cependant, comme expliqué précédemment, ce nuage a été divisé par la suite en deux sous-nuages afin d'analyser d'un côté le bâtiment entier et de l'autre les bâtiments incomplets. Ces deux sous-nuages sont repris aux figures 14 et 15. Le nuage contenant uniquement le bâtiment est plus petit puisqu'il compte un peu moins de 36 millions de points alors que le second en compte un peu plus de 89 millions

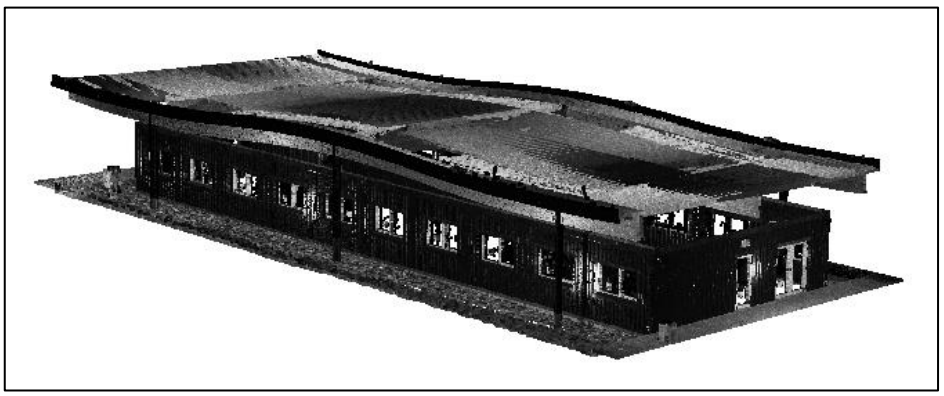


Figure 14 : Sous-nuage n°1 (35 786 930 points)

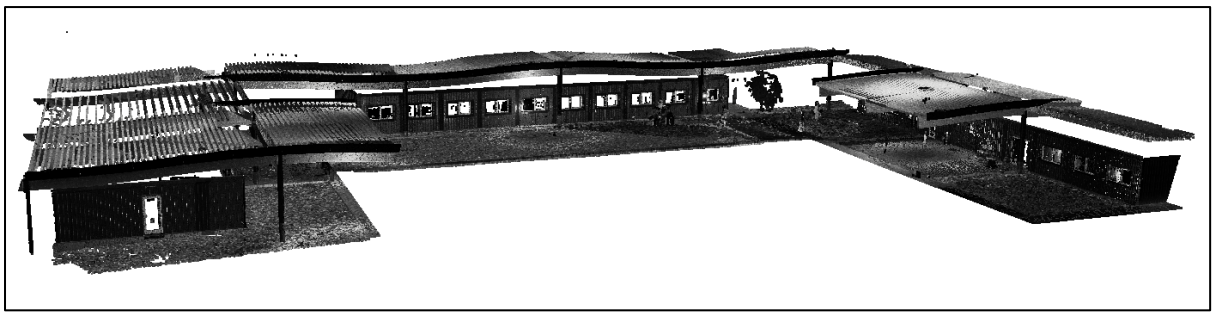


Figure 15 : Sous-nuage n°2 (89 213 070 points)

Outre le nuage de points, un fichier texte contenant les coordonnées des 54 cibles a été exporté du logiciel *Cyclone*. Manuellement, les codes associés à chaque cible ont été ajoutés dans ce fichier de manière à obtenir un résultat complet et utilisable pour la segmentation. Le fichier contenant l'identifiant, les coordonnées X, Y et Z et le code de chaque cible est repris à l'annexe 10.4.

6.1.3. Segmentation et classification

L'algorithme de segmentation et classification, pour cette méthode comme pour l'autre, n'a pas été développé sur le nuage total. En effet, afin de limiter les temps de traitement lors des différents tests et de pouvoir valider les méthodes, un nuage échantillonné reprenant uniquement deux façades du bâtiment principal a été utilisé pour les développements. Celui-ci contient 500 000 points au lieu des 36 millions du sous-nuage n°1. Il est repris à la figure 16. L'algorithme, repris à l'annexe 10.1, a ensuite été appliqué à la totalité des deux sous-nuages pour tester son efficacité et comparer les résultats avec ceux de la segmentation manuelle (cf. 6.3). Les résultats repris dans cette partie sont ceux obtenus sur les sous-nuages non-échantillonnés afin de faciliter la visualisation.

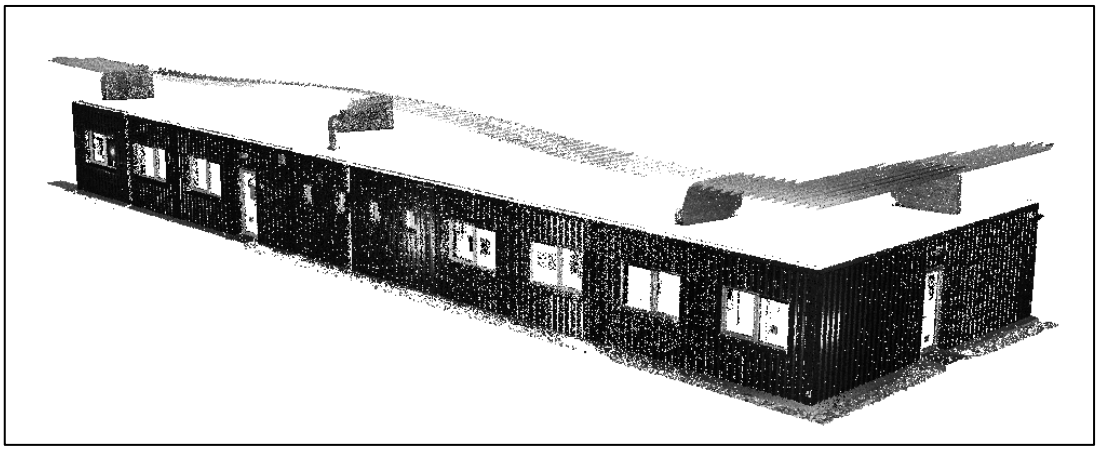


Figure 16 : Nuage utilisé pour le développement des algorithmes (500 000 points)

Les résultats sont d'abord présentés pour chaque classe d'objets en suivant la méthodologie détaillée au point 4.1.4 et en expliquant le choix des différents seuils. Les deux sous-nuages classifiés sont repris à la fin de cette partie aux figures 23 et 24.

6.1.3.1. Murs

Pour cette méthode, l'algorithme de segmentation commence avec l'extraction des murs. Avant d'appliquer RANSAC pour extraire le plan qui correspond le mieux au mur, un plan initial est calculé sur base du voisinage de la cible. Le rayon de recherche de ce voisinage est fixé à 5 cm puisque cela correspond à la dimension des cibles de part et d'autre de leur centre. C'est le même rayon qui sera utilisé pour déterminer le voisinage des cibles associées aux portes et aux fenêtres. En effet, ces cibles sont toutes de taille identique (seules celles posées sur les gouttières sont plus petites).

Pour limiter le nombre d'itérations et donc les temps de traitements lors de la recherche du meilleur plan, les points testés sont uniquement ceux situés à moins de 1 mètre du plan initial. Cela permet de réduire grandement la quantité de points pour lesquels la distance au plan doit être calculée. De plus, l'angle entre la normale du plan testé et la normale du plan initial doit être inférieur à $\pi/20$. Ainsi, ces deux conditions permettent de limiter le temps de recherche du meilleur plan tout en augmentant la probabilité de trouver le plan adéquat. 30 itérations sont effectuées pour extraire le meilleur plan correspondant à chaque mur.

Il reste toutefois un paramètre essentiel à choisir pour extraire le meilleur plan : la distance maximale des points à celui-ci. Pour définir le seuil le plus adapté, plusieurs essais ont été effectués. En effet, une distance trop petite conduit à une sous-segmentation du mur, surtout avec des murs en tôle ondulée comme c'est le cas ici. A l'inverse, une distance trop grande engendre la prise en compte de points des portes et du sol dans la classe « Murs ». Il faut donc choisir un seuil intermédiaire limitant l'attribution de points à de mauvaises classes. Une distance au plan de 6 cm semble fournir un résultat convenable. La figure 17 présente les résultats obtenus pour un mur avec différents seuils de distance.

Il faut ensuite supprimer les points situés au-dessus des murs. Pour cela, l'histogramme des altitudes est déterminé. Celui obtenu pour le sous-nuage n°1 est repris à la figure 18. Il met en évidence la séparation entre les points appartenant aux murs et les points situés au-dessus. De cette manière, les points compris dans les plans mais dont l'altitude est supérieure à celle de la dernière tranche du mur peuvent être exclus.

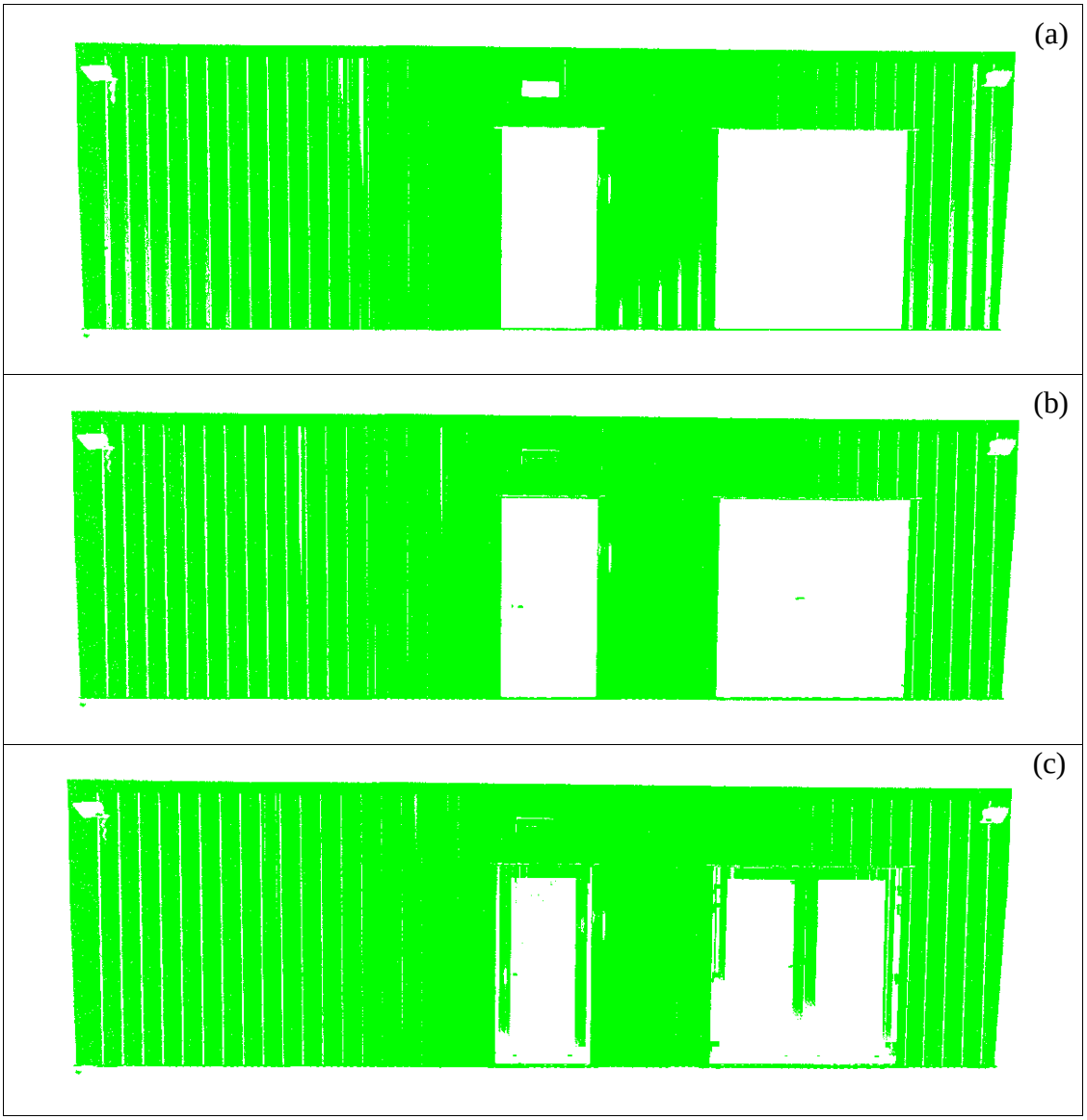


Figure 17 : Segmentation d'un mur par la méthode des cibles

(a) Seuil = 3 cm ; (b) Seuil = 6 cm ; (c) Seuil = 10 cm

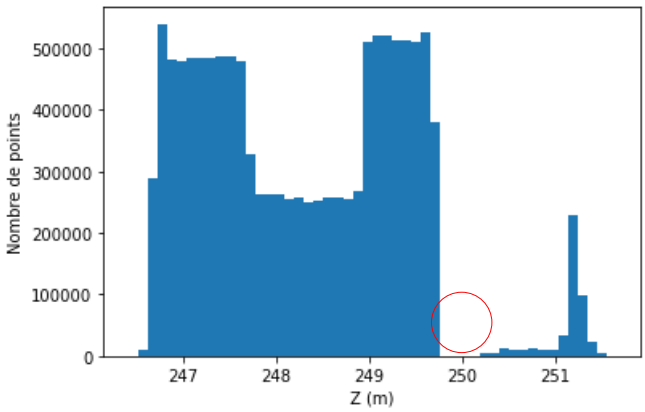


Figure 18 : Histogramme des altitudes des points inclus dans les plans des murs extraits du sous-nuage n°1

Les points situés dans les prolongements des murs doivent également être retirés. Pour cela, une grille 2D de pixels de 5 cm a été établie pour l'ensemble des points repris en tant que murs. En considérant qu'un mur fait minimum deux mètres de haut, les pixels dont la différence entre l'altitude minimale et l'altitude maximale des points est inférieure à 2 m sont exclus. Une résolution de 5 cm peut ne pas être suffisamment discriminante pour éliminer les points situés aux marges des bâtiments mais utiliser une résolution plus fine pourrait poser problème là où la densité de points est faible. La figure suivante illustre les différentes étapes de l'extraction des murs. Les points appartenant à la classe « Murs » sont ensuite mis de côté pour continuer l'algorithme sur le reste des données.

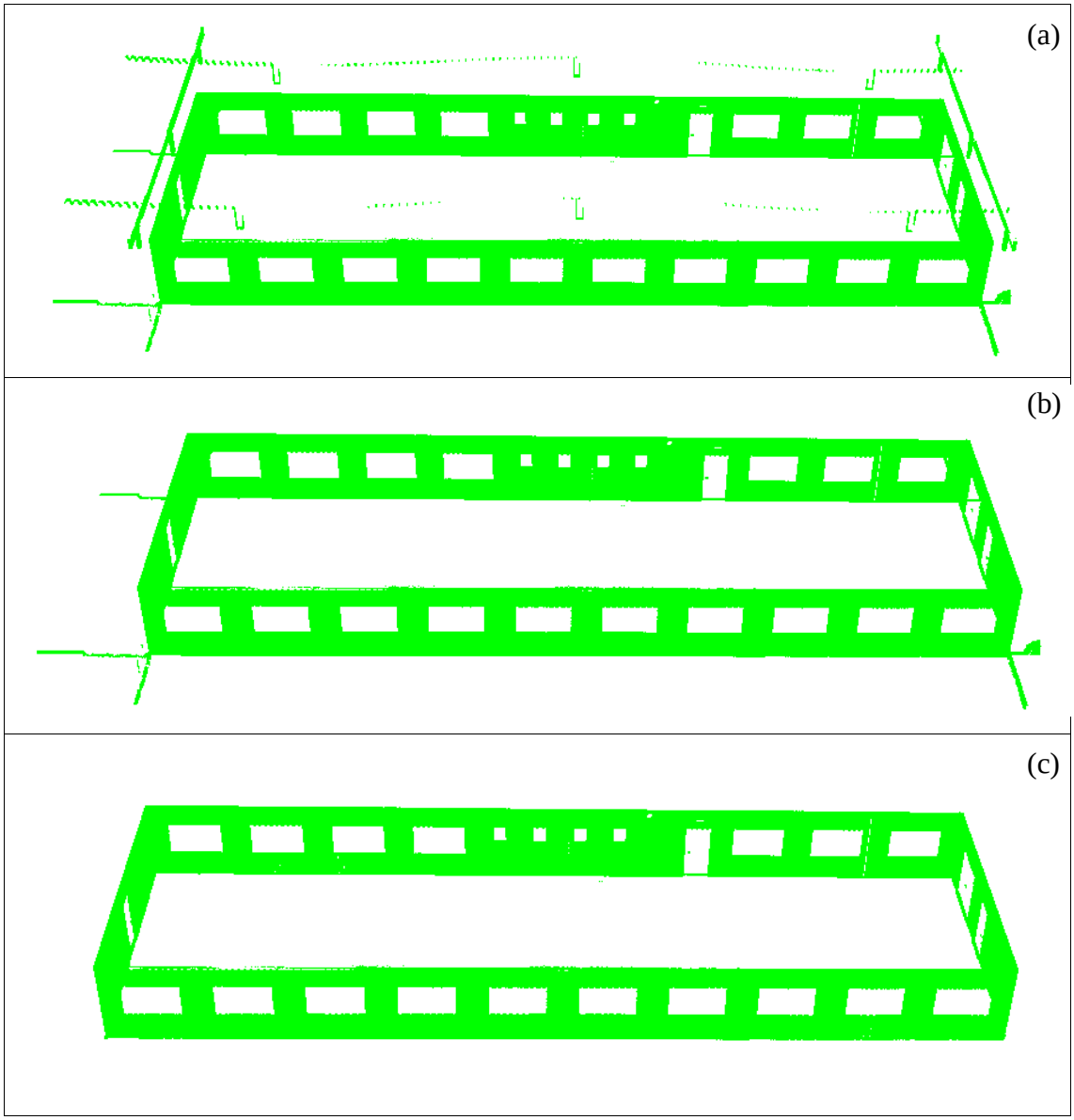


Figure 19 : Extraction des murs par la méthode des cibles

(a) Extraction des plans ; (b) Elimination des points au-dessus ; (c) Elimination des points sur les côtés pour obtenir le résultat final

La segmentation des murs prend du temps puisqu'il faut en moyenne 40 minutes pour les extraire du sous-nuage n°1. Cela s'explique par le fait que toute une série de plans doivent être testés en calculant à chaque fois la distance de nombreux points du nuage à ce plan. Le sous-nuage n°2 comprenant plus de points, le temps d'extraction des murs est plus long. Il tourne aux alentours des 60-65 minutes.

6.1.3.2. Portes

Une fois les murs extraits, la segmentation des portes peut commencer. Celle-ci est réalisée sur le reste du nuage en déterminant le plan qui s'ajuste le mieux au voisinage de la cible (rayon de recherche de 5 cm). Il faut ensuite déterminer la distance au plan à partir de laquelle les points sont attribués à la classe « Portes » pour tenir compte de l'épaisseur des portes et de la précision du plan. A nouveau, ce seuil de distance est déterminé de manière empirique afin de trouver celui qui est le plus adapté. La figure 20 présente trois cas. Le premier cas montre l'impact d'un seuil trop petit, engendrant une sous-segmentation de la porte. Le second cas présente le seuil semblant fournir le meilleur résultat : 4 cm. Le dernier cas illustre l'utilisation d'un seuil de distance trop grand ce qui entraine la prise en compte de points extérieurs à l'objet (mur et sol). En outre, afin de limiter l'extension du plan à la porte, les points appartenant à cette classe doivent remplir une deuxième condition, ils doivent être situés à moins de 2 m du centre de la cible. Sans cette condition, des fenêtres alignées avec la porte se retrouveraient également dans cette classe.

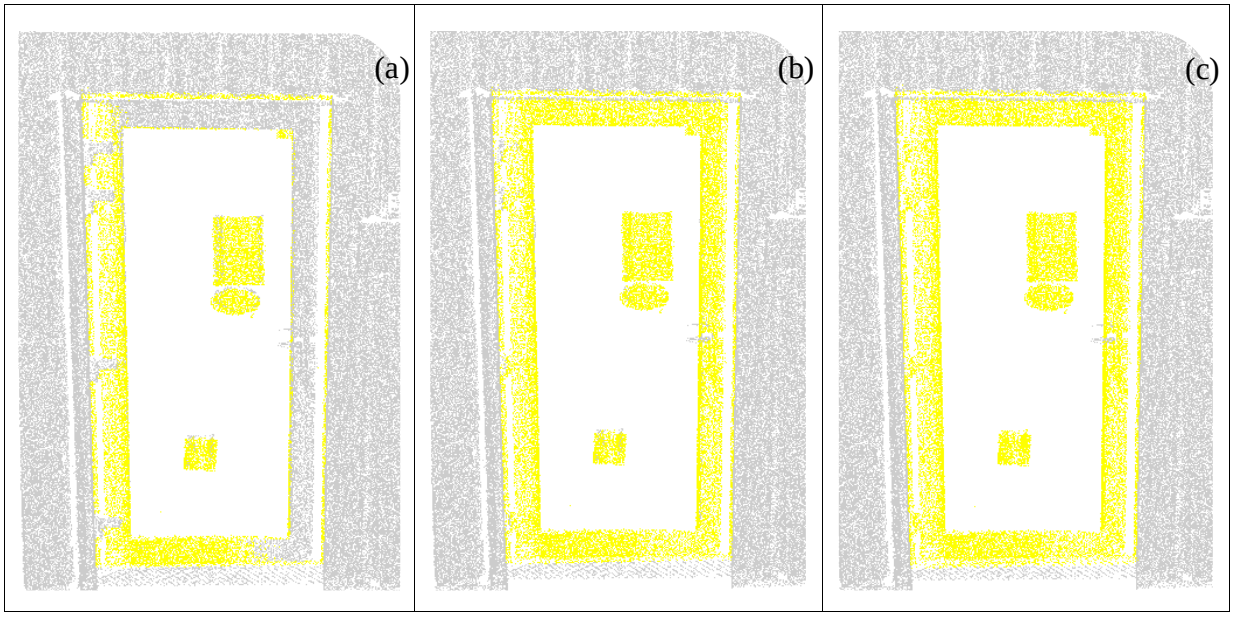


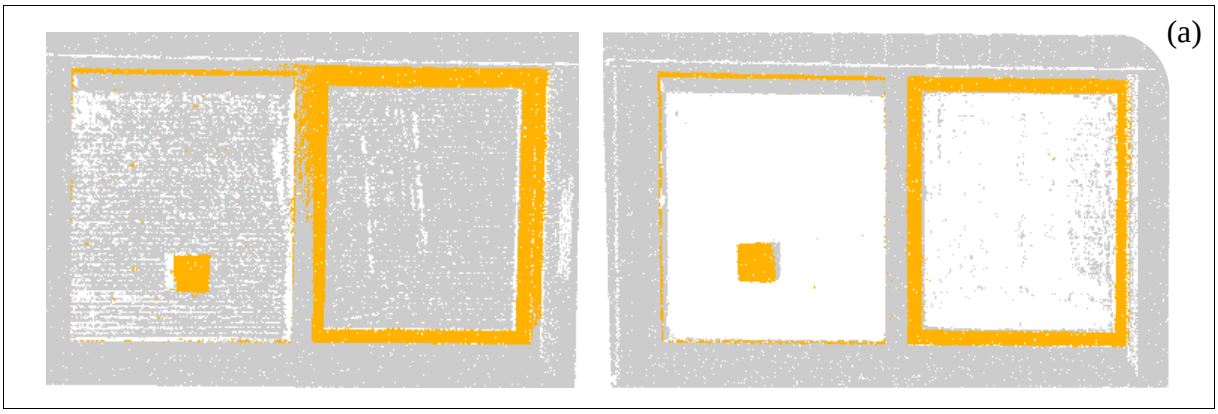
Figure 20 : Segmentation d'une porte par la méthode des cibles
(a) Seuil = 2 cm ; (b) Seuil = 4 cm ; (c) Seuil = 7 cm

Il faut environ 300 secondes pour segmenter les portes du sous-nuage n°1. La taille du nuage de points joue encore un rôle prépondérant dans les temps de traitement puisque l'extraction prend en moyenne 600 secondes pour le sous-nuage n°2.

6.1.3.3. Fenêtres

Pour les fenêtres, le principe de segmentation est similaire à celui des portes. Néanmoins, au lieu d'utiliser une seule cible, le plan correspondant aux fenêtres d'une façade est déterminé à partir des deux cibles (et de leurs points voisins) situées aux extrémités de la façade. Il faut ensuite déterminer la distance au plan définissant quels points appartiennent aux fenêtres. Les mêmes tests que précédemment ont été effectués en appliquant différents seuils et en comparant les résultats. La figure 21 reprend les résultats obtenus avec trois seuils différents. Le résultat présenté au milieu correspond au seuil choisi. Comme pour les portes, ce sont les points situés à moins de 4 cm du plan qui sont pris en compte. Ce seuil est à appliquer à l'ensemble du nuage. Avec une distance plus petite, une part importante du châssis n'est pas intégrée dans la classe. Avec une distance plus grande, une partie du bruit se retrouve associée à la classe « Fenêtres ». Il faut de nouveau limiter l'extension du plan pour éviter l'extraction de points appartenant au sol ou au toit. Pour cela, une marge de 1 m est appliquée aux coordonnées minimales et maximales des cibles afin de limiter la zone candidate. Idéalement, pour plus de précision, une marge plus petite serait recommandée mais la position des cibles ne le permet pas. En effet, celles-ci ne sont pas situées suffisamment aux extrémités des fenêtres pour pouvoir diminuer cette marge. Ce point sera abordé dans la discussion.

La segmentation des fenêtres se fait approximativement en 150 secondes pour le sous-nuage n°1. Elle est nettement plus longue pour le sous-nuage n°2 puisqu'elle prend en moyenne 650 secondes.



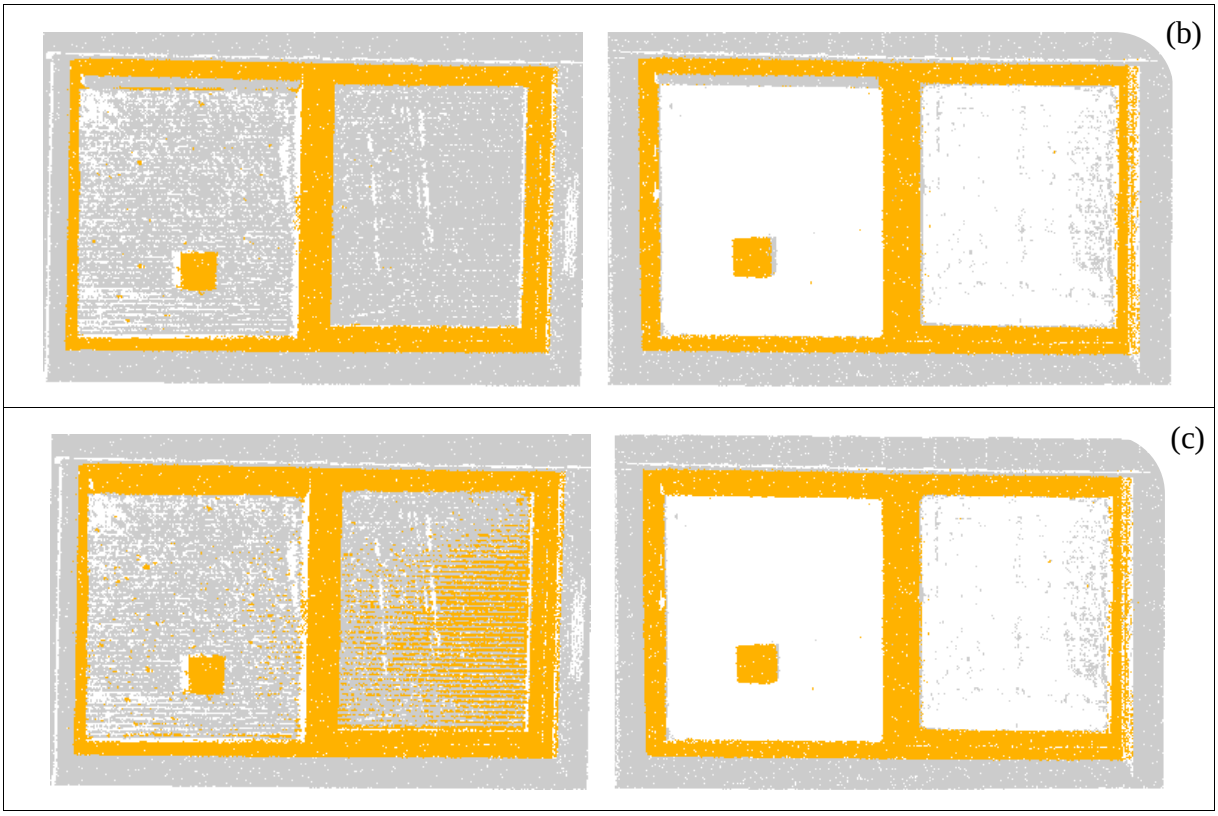


Figure 21 : Segmentation des fenêtres par la méthode des cibles
(a) Seuil = 2 cm ; (b) Seuil = 4 cm ; (c) Seuil = 6 cm

6.1.3.4. Gouttières

Après la segmentation des murs, portes et fenêtres, il reste à extraire les gouttières. Pour cela, une croissance de région est appliquée au départ de chaque cible associée à une gouttière. Au préalable, sachant que le diamètre d'une gouttière n'est que de quelques centimètres, les points situés à une distance horizontale de moins de 20 cm d'une cible sont extraits. Cela permet de travailler sur un nuage de points beaucoup plus petit et ainsi limiter les temps de traitement. Les normales de chaque point du petit nuage sont ensuite calculées afin de pouvoir effectuer la croissance de région sur base de cette caractéristique. Au départ, l'idée était également de prendre en compte la courbure comme critère mais la répartition irrégulière des points n'en permettait pas une détermination fiable. Chaque normale est calculée sur base des voisins situés à moins de 3 cm du point.

Les critères de croissance de région ont été déterminés empiriquement mais les résultats fournis pourraient sans doute être améliorés. Le premier est un critère de distance : deux points sont voisins s'ils sont situés à moins de 2 cm l'un de l'autre. Une plus petite distance pourrait permettre de limiter la croissance aux points appartenant réellement à la gouttière mais la densité de points est trop faible par endroit pour pouvoir diminuer ce seuil. Le

deuxième critère est défini sur base de l'angle formé par les normales de deux points successifs. L'angle doit être inférieur à $\pi/10$ pour que les deux points appartiennent à la même région. Enfin, le troisième critère est rempli si la composante en Z du vecteur normal testé est inférieure à 0,1 (c'est-à-dire que le vecteur est proche de l'horizontale).

Plusieurs problèmes illustrés à la figure 22 n'ont néanmoins pas pu être éliminés par ces critères. En effet, le changement entre gouttière et mur n'est pas assez franc à l'extrémité supérieure des gouttières ce qui engendre la prise en compte de points extérieurs. C'est le problème principal. Celui-ci est également présent à l'extrémité inférieure de la deuxième gouttière. Ensuite, un problème spécifique à la première gouttière conduit à une mauvaise segmentation de celle-ci. De fait, elle se situe dans le prolongement d'un mur ce qui fait qu'elle se retrouve principalement dans cette classe et non dans la classe « Gouttières ».

Outre ces problèmes dans la méthode de segmentation, les deux cibles placées sur les gouttières les plus éloignées de la zone n'ont pu être reconnues par le logiciel (densité de points trop faible). Par conséquent, sur les six gouttières présentes, seules quatre ont pu être extraites du nuage ce qui impacte évidemment la qualité des résultats. Les gouttières manquantes sont également mises en évidence sur la figure ci-dessous.

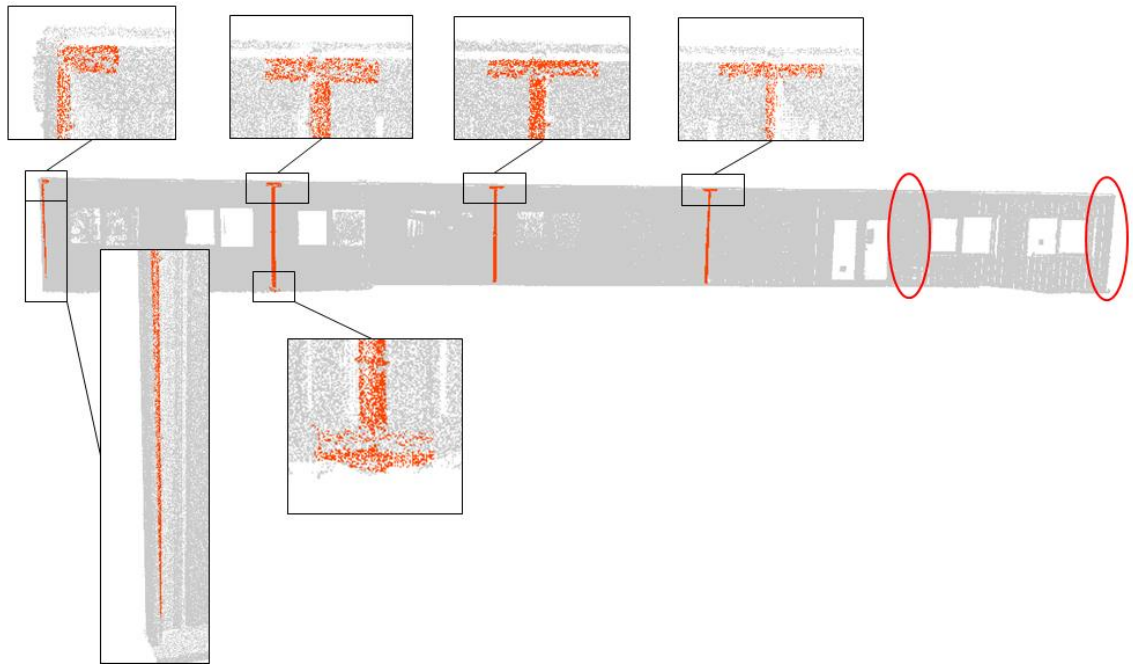


Figure 22 : Mise en évidence des problèmes de la segmentation des gouttières (méthode des cibles)

La segmentation des gouttières n'est valable que pour le sous-nuage n°2 (comprenant les bâtiments incomplets) puisqu'il n'y en a pas sur l'autre bâtiment. Le temps d'extraction des gouttières pour ce nuage avoisine les 50 secondes.

6.1.3.5. Résultat final

La figure ci-dessous reprend le sous-nuage n°1 classifié. Celui-ci ne comporte que quatre classes puisque le bâtiment ne dispose pas de gouttières. La classe reprenant le bruit et les autres éléments faisant partie du nuage est reprise en bleu. Les murs sont en vert, les portes en jaune et les fenêtres en orange.

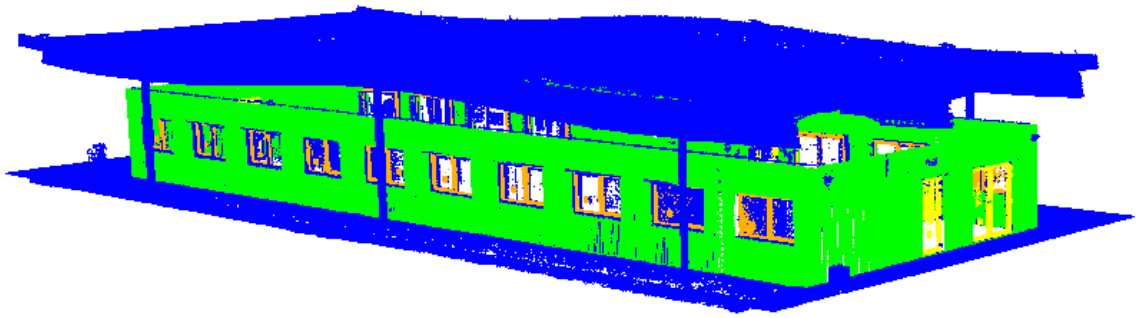


Figure 23 : Sous-nuage n°1 classifié par la méthode des cibles

Le sous-nuage n°2, repris à la figure ci-dessous, comporte une cinquième classe puisque quelques gouttières sont présentes (en rouge). On peut voir que le morceau de mur en bas à droite n'est pas repris dans la classe « Murs ». Cela s'explique par le fait qu'aucune cible n'avait encore été placée dessus. Il ne faisait donc pas partie des murs à segmenter.

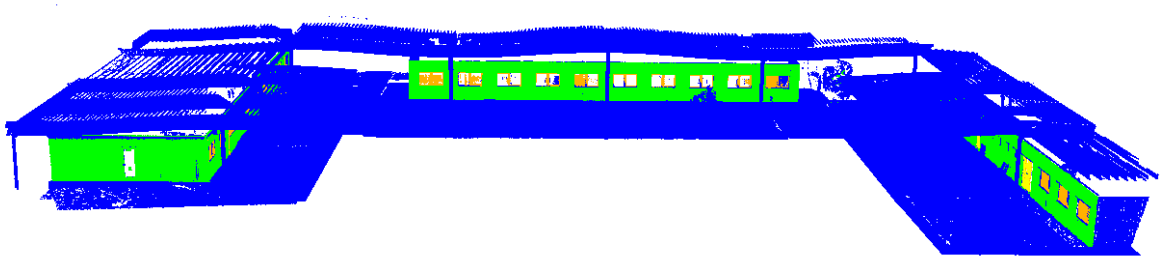


Figure 24 : Sous-nuage n°2 classifié par la méthode des cibles

Afin de faciliter la visualisation, la figure suivante regroupe les deux sous-nuages en masquant la classe 0 contenant notamment le bruit. Plusieurs problèmes se remarquent, principalement pour les bâtiments incomplets, tels que la présence d'une partie importante du bruit dans la classe « Fenêtres ». En outre, les deux portes du bâtiment avec les gouttières ne sont pas segmentées. Ce souci provient du fait que, contrairement aux portes présentes lors du développement, celles-ci ne sont pas renfoncées. Elles sont donc comprises dans le plan du mur et se voient attribuer à cette classe. Cette éventualité n'avait pas été envisagée lors de la création du script. Les résultats s'en trouvent donc affectés.

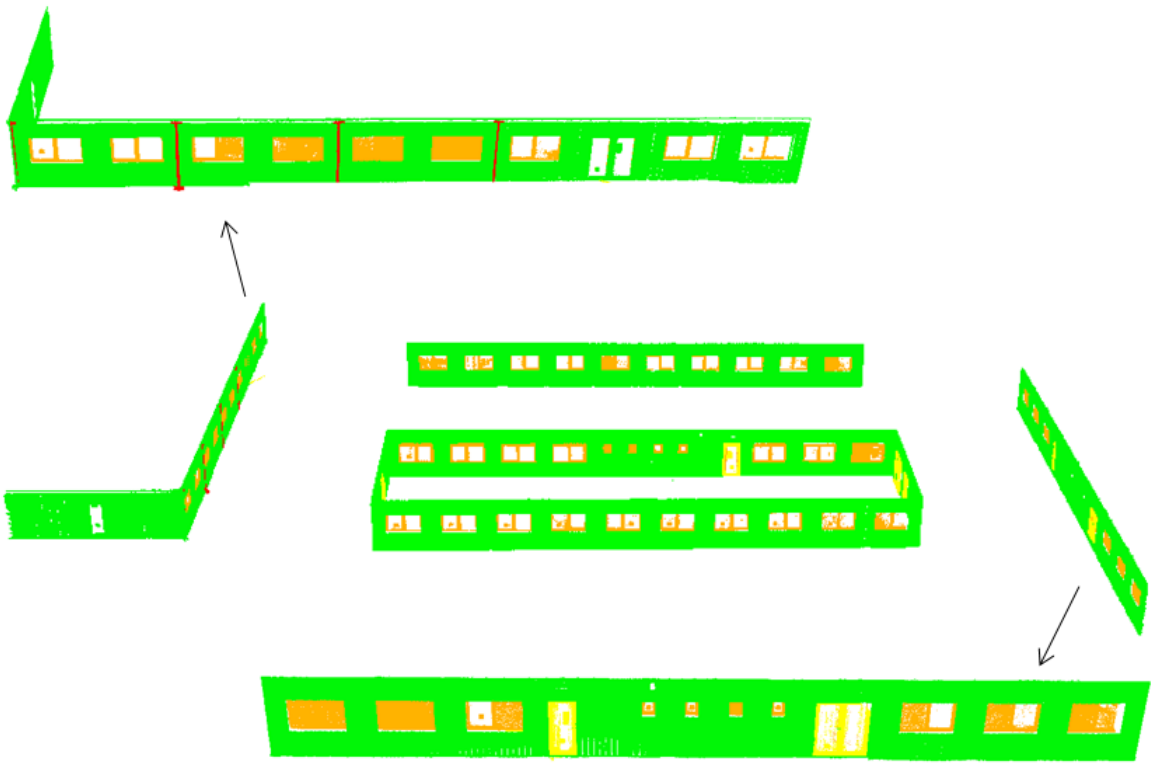


Figure 25 : Résultat final de la classification issue de la méthode des cibles (en excluant la classe 0)

La durée totale de l'algorithme de segmentation/classification pour cette méthode est d'un peu moins de 50 minutes pour le sous-nuage n°1. Pour le sous-nuage n°2, elle est plus longue puisqu'il faut plus de 80 minutes. Cet écart provient principalement de la différence de taille des deux nuages de points. La répartition des temps de traitements est présentée sur le graphique ci-dessous.

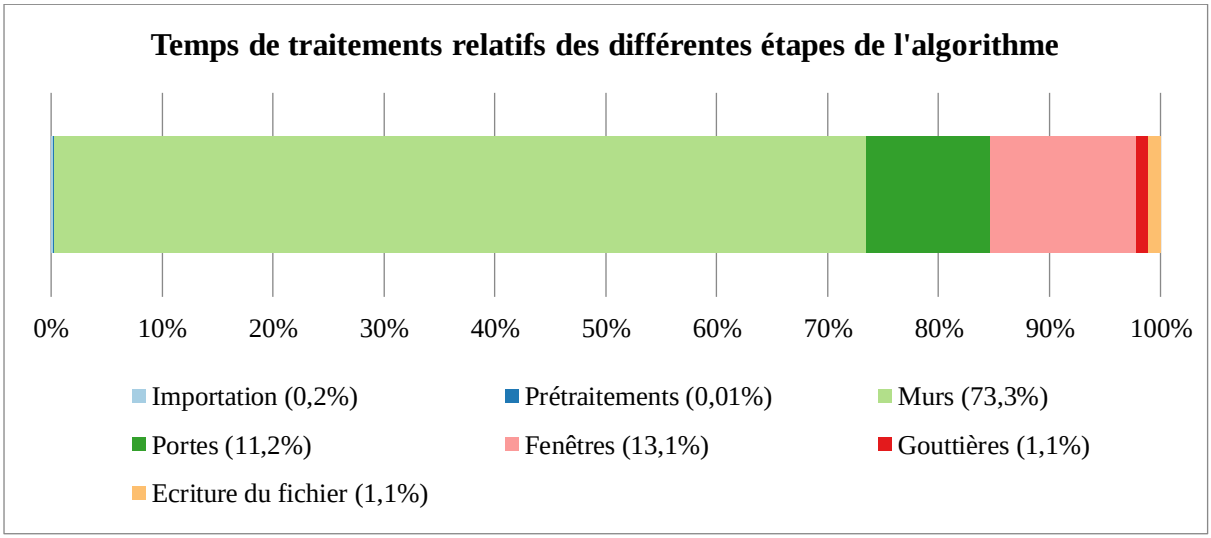


Figure 26 : Temps de traitements relatifs des étapes de l'algorithme basé sur les cibles

On peut voir que c'est l'extraction des murs qui prend le plus de temps avec plus de 70% de la durée totale de l'algorithme. La segmentation des fenêtres et des portes suivent plus loin avec respectivement 13% et 11% de la durée totale. Les autres étapes de l'algorithme ne représentent dès lors qu'une infime part du temps total.

6.2. Méthode par levé topographique

6.2.1. Géocodification

La géocodification utilisée pour cette méthode est similaire à celle utilisée par les géomètres lors de leurs levés. Néanmoins, le nombre d'objets d'intérêt étant limité, celle-ci est plus réduite. Il serait évidemment possible d'ajouter d'autres codes pour étendre cette technique à d'autres éléments mais cela ne rentre pas dans le cadre de ce travail. Le tableau ci-dessous reprend les codes utilisés pendant le levé, inspirés d'une géocodification établie lors d'un projet réalisé auparavant. Les explications relatives aux codes se trouvent dans le chapitre consacré à la méthodologie (cf. 4.2.1.).

Objet	Description	Code	
		Séquence	Modificateur
Bâtiment	Contour supérieur + 1 point inférieur	100 - 105	.2 (début) .4 (suite)
Porte	2 points extrêmes	350	
	3 points car un coin non-visible	351	.8 (point décalé)
	Contour complexe	355	.2 (début) .4 (suite)
Fenêtre	2 points extrêmes	360	
	3 points car un coin non-visible	361	.8 (point décalé)
	Contour complexe	365	.2 (début) .4 (suite)
Gouttière	Centre	408	
	Rayon	409	
	Hauteur	410	
Station	Visées entre stations	921	
Point de référence	Tour d'horizon	999	

Tableau 3 : Géocodification utilisée pour le levé topographique

Contrairement à la méthode des cibles, les codes utilisés dans ce cas-ci sont différents des valeurs utilisées pour la classification (comprises entre 0 et 4). Cela permet de mesurer de plusieurs manières un même type d'objet tel que les portes et les fenêtres ou encore de lever plusieurs bâtiments en même temps, en utilisant des séquences distinctes.

6.2.2. Acquisition et prétraitements

Le levé topographique a été fait avant l'acquisition au laserscan. Il a donc été effectué sur la totalité de la zone initialement prévue. A partir de cinq clous de coordonnées connues présents dans la zone, une polygonale a été établie en ajoutant six nouvelles stations afin de pouvoir effectuer la totalité des mesures. La polygonale en réseau, maximisant les inter-visées, est illustrée à la figure 27. Les points déjà présents au début des mesures y sont repris en rouge alors que les nouveaux points implantés sont en jaune. Le système de référence utilisé est celui des points existants : le Lambert 2008. Pour extraire les bâtiments, portes, fenêtres et gouttières, le nombre de points levés sur l'intégralité de la zone s'élève à 437.

Deux jours de travail ont été nécessaires pour effectuer le levé topographique. Néanmoins, comme la zone a été restreinte, une partie conséquente des points levés n'est plus utile. La durée d'acquisition peut donc approximativement être divisée par deux (≈ 7h).

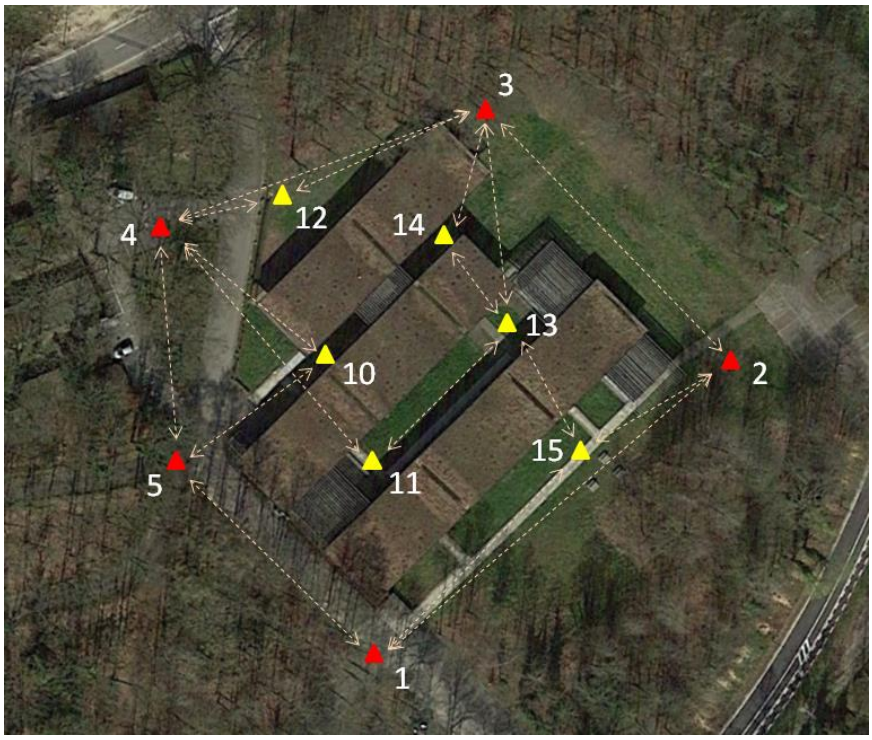


Figure 27 : Schéma de la polygonale reprenant les positions des stations et les inter-visées

Les coordonnées des nouvelles stations et des points levés ont été calculées avec le logiciel *Covadis*. En appliquant le principe des moindres carrés, l'erreur quadratique moyenne sur les coordonnées ajustées des stations oscille entre 1 et 3 mm en planimétrie et entre 1 et 2 mm en altimétrie. Plus l'erreur sur une station est grande, plus les mesures effectuées depuis celle-ci sont inexactes. Pour que la méthode soit efficace, il est impératif de limiter les imprécisions sur les points levés pour qu'ils soient positionnés correctement par rapport au nuage de points. Le géoréférencement s'est fait directement puisque les deux jeux de données sont exprimés dans le même système.

La liste reprenant la totalité des points de stations (code 921) et des points levés est reprise à l'annexe 10.5.

6.2.3. Segmentation et classification

Pour les mêmes raisons que celles expliquées précédemment, l'algorithme de segmentation a été développé sur un nuage plus petit (figure 16). Il a ensuite été appliqué sur l'ensemble des deux sous-nuages pour lesquels les résultats sont présentés ci-dessous. L'annexe 10.2 reprend le script développé pour cette seconde méthode. Les résultats sont expliqués pour chaque classe d'objet, dans l'ordre de leur extraction et en suivant la méthodologie expliquée au point 4.2.4. La classification finale est reprise à la fin de cette partie aux figures 34 et 35.

6.2.3.1. Gouttières

Les gouttières sont les premiers éléments analysés pour cette méthode. En effet, comme les bâtiments sont extraits en entier et que les gouttières sont situées à la limite entre bâtiment et extérieur, il est préférable d'effectuer leur extraction sur l'intégralité du nuage et non juste sur les points du bâtiment. La méthode de segmentation des gouttières se base sur une extraction cylindrique (cf. 4.2.4.3.). A partir des trois points de coordonnées connues (centre, rayon et hauteur), chaque gouttière peut être extraite. Il faut néanmoins déterminer la marge de sécurité à appliquer pour tenir compte de la précision du levé.

Cette marge de sécurité est déterminée de manière empirique en testant différentes valeurs et en comparant les résultats avec la segmentation manuelle d'une gouttière. La figure suivante illustre les résultats obtenus pour différents seuils. On peut voir que le seuil de 1 cm est trop discriminant alors que le seuil de 5 cm ne l'est pas assez. Le seuil de 2 cm fournit le meilleur résultat pour la gouttière présentée, même si la segmentation n'est pas parfaite. C'est donc celui-là qui sera appliqué sur l'ensemble des gouttières.

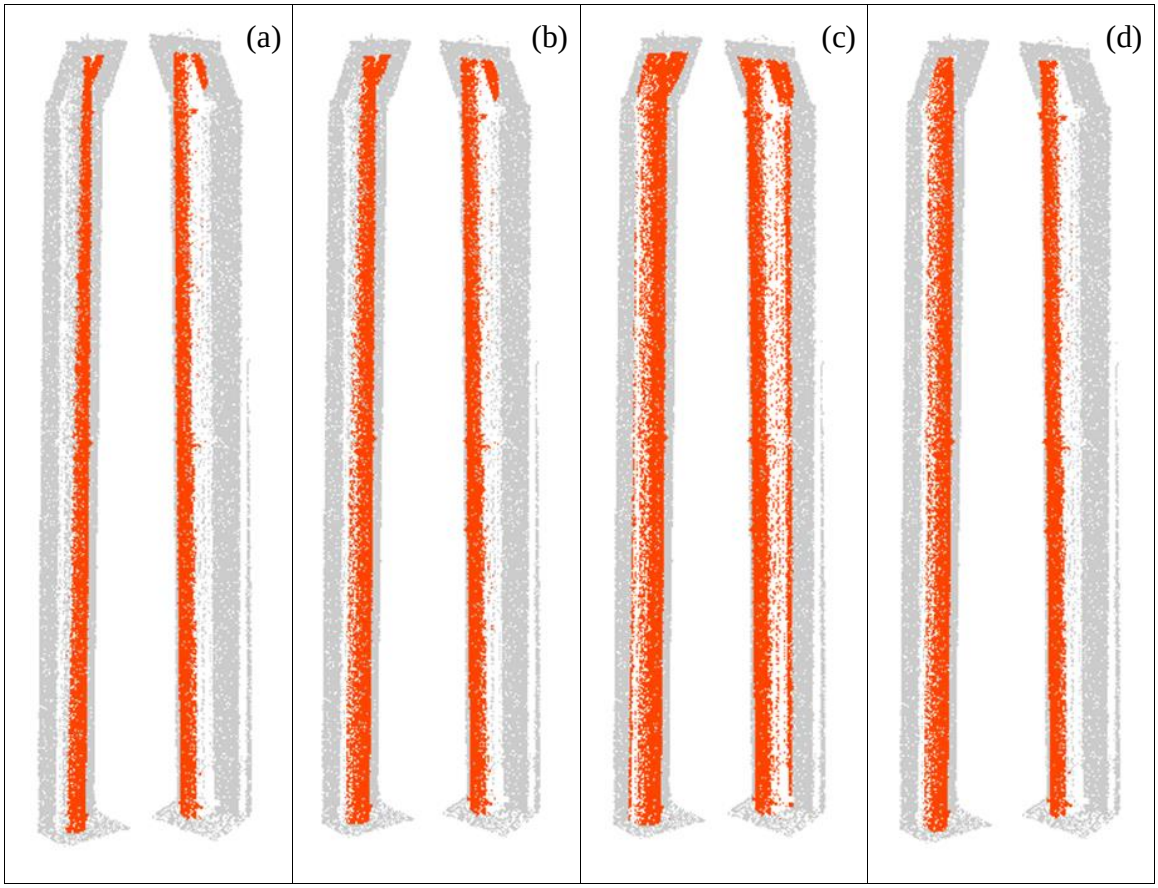


Figure 28 : Segmentation d'une gouttière par la méthode avec levé topographique
(a) Seuil = 1 cm ; (b) Seuil = 2 cm ; (c) Seuil = 5 cm ; (d) Segmentation manuelle

Plusieurs problèmes, illustrés à la figure 29, sont néanmoins à mettre en évidence. Le problème n°1 vient du fait que des points du mur se retrouvent dans la classe « Gouttières ». Cette situation pourrait provenir du choix d'une marge de sécurité trop grande. Cependant, la figure précédente montre qu'utiliser un seuil plus petit conduirait au non-classement de nombreux points. La proximité entre le mur et les gouttières rend leur distinction difficile, d'autant plus qu'il faut tenir compte de la précision des points levés. Celle-ci est d'ailleurs à l'origine du problème n°2. En effet, une gouttière est particulièrement mal segmentée avec peu de points attribués à la classe « Gouttières ». Comme ce n'est pas le cas pour les autres, l'explication la plus logique serait un manque de précision lors de l'acquisition de cette gouttière avec, sans doute, un mauvais positionnement du prisme. Enfin, le troisième problème mis en évidence provient de l'inclinaison du mur au niveau de l'entrée de la gouttière. Ce changement engendre l'affectation à mauvaise classe des points situés sur cette pente et qui se retrouvent donc plus proches du centre de la gouttière.

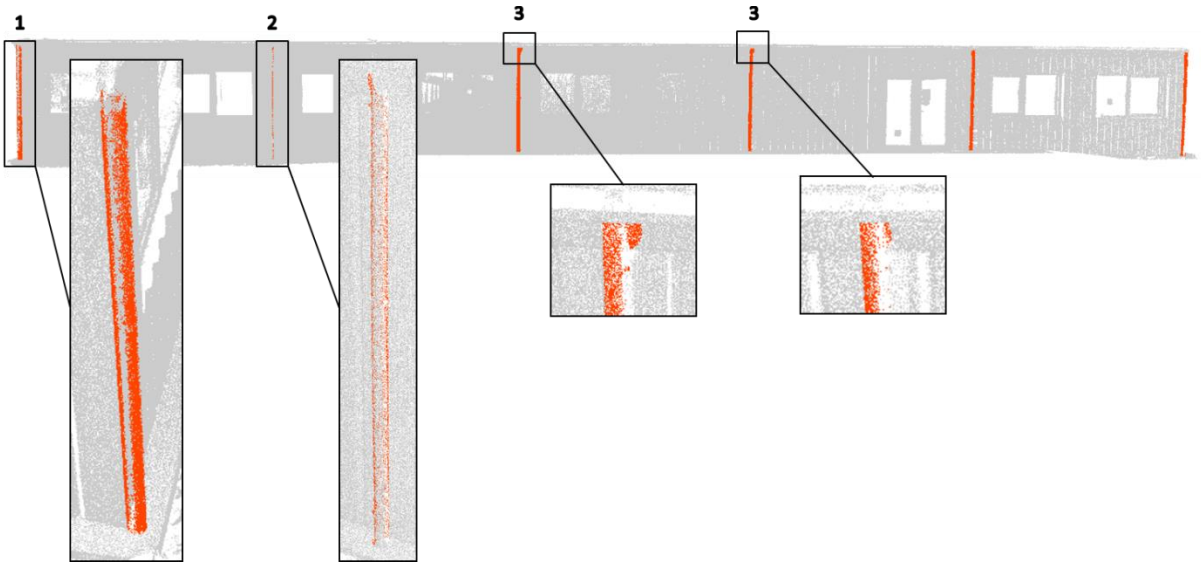


Figure 29 : Mise en évidence des problèmes de la segmentation des gouttières (méthode du levé)

La segmentation des gouttières est seulement utile pour le sous-nuage n°2 puisque l'autre sous-nuage n'en contient pas. Le temps de traitement moyen pour extraire les gouttières de ce sous-nuage est de 24 secondes.

6.2.3.2. Bâtiment/murs

Après avoir retiré les éventuelles gouttières du nuage de points, la fonction *Bounding Box 3D* (cf. 4.2.4.1.) a été appliquée pour extraire les bâtiments. Contrairement à la méthode précédente, ce ne sont pas juste les murs qui sont segmentés. Le bâtiment entier est extrait comme un sous-nuage et c'est à partir de celui-ci que les portes et fenêtres seront à leur tour segmentées. Les murs seront donc les points restants de ce sous-nuage particulier. Dès lors, le bruit à l'intérieur du bâtiment doit être éliminé sinon il se retrouverait dans cette classe-ci. A cette fin, la fonction est appliquée deux fois. La première afin d'extraire tout le bâtiment, la seconde afin de retirer la majorité du bruit intérieur.

La fonction *Bounding Box 3D* se base sur la fonction *path.contains_points* de la librairie Matplotlib pour extraire les points contenus à l'intérieur du polygone formé par les points levés. Celle-ci dispose d'un paramètre nommé *radius* qui permet d'agrandir (ou de rétrécir) le polygone de la moitié de la valeur fournie. Le choix de la valeur de ce paramètre est donc essentiel pour extraire correctement le bâtiment. Celui-ci a été fixé de manière empirique tout en tenant compte de la précision des points et de l'épaisseur des murs. Ainsi, après plusieurs essais, le paramètre a été fixé à 0,08 pour appliquer la première fonction. Cela signifie que l'emprise du bâtiment est agrandie de 4 cm. L'application de valeurs plus grandes impliquait

la prise en compte d'une partie importante du sol au pied des bâtiments. A l'inverse, utiliser des valeurs plus petites engendrait la suppression de points appartenant aux murs. Ces trois cas sont illustrés pour un des murs à la figure suivante. Ensuite, pour retirer les points à l'intérieur du bâtiment, cette valeur a été multipliée par 6 afin de rétrécir l'emprise de 24 cm, permettant ainsi de conserver les murs et les châssis dans le nuage de points. En diminuant encore l'emprise (marge > 24 cm), une grande partie du bruit derrière les vitres est prise en compte. Ce bruit se verrait donc attribué à la mauvaise classe, affectant la qualité des résultats. Par contre, en conservant moins de points (marge < 24 cm), c'est la qualité des fenêtres qui est impactée puisque certains points de cette classe en seraient directement exclus. En outre, une marge de sécurité de 2 cm est appliquée aux altitudes minimales et maximales des points.

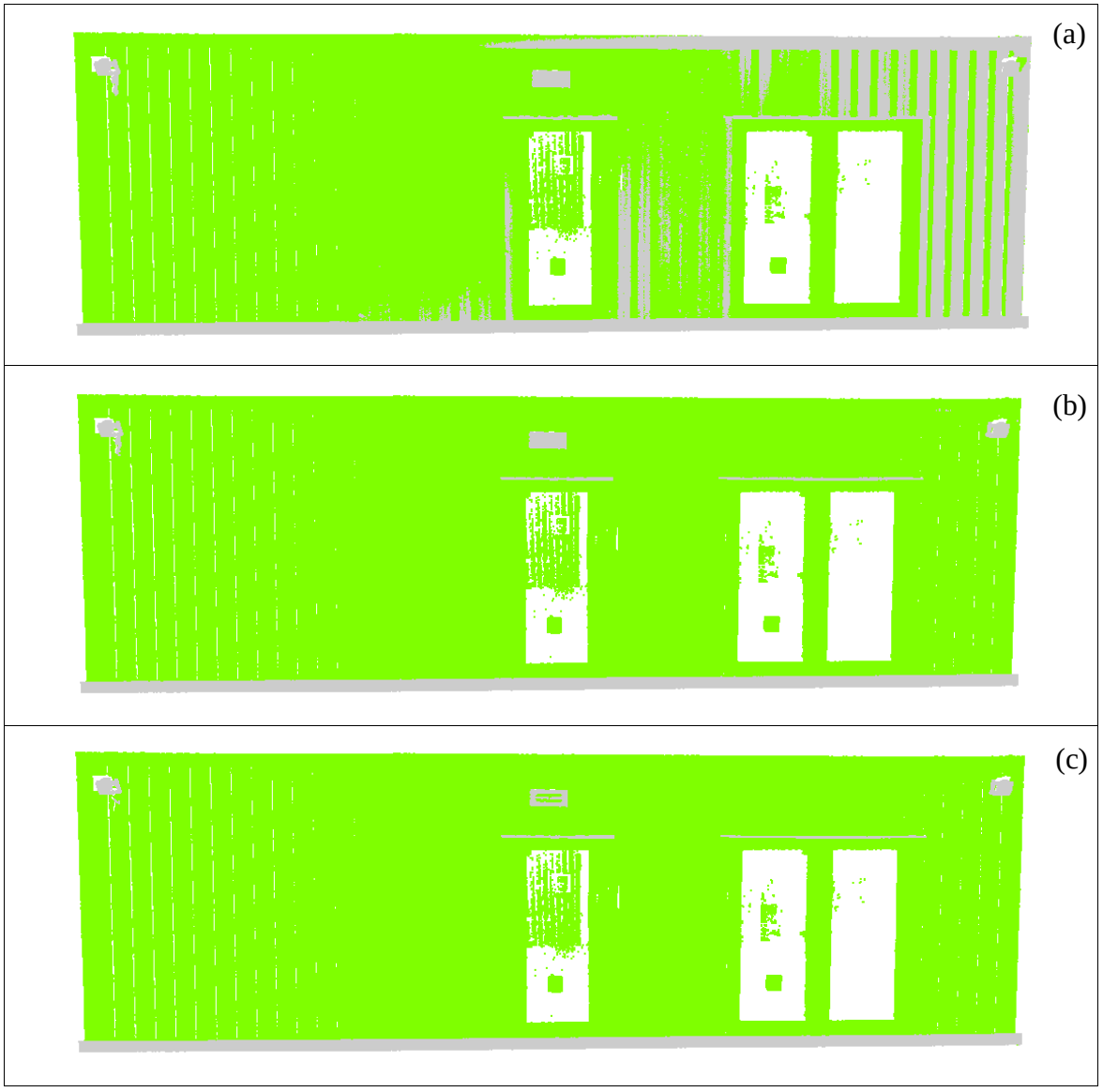


Figure 30 : Segmentation d'un mur par la méthode avec levé topographique
(a) Seuil = 2 cm ; (b) Seuil = 4 cm ; (c) Seuil = 6 cm

Au niveau des temps de traitements, il faut en moyenne 15 secondes pour extraire le bâtiment du sous-nuage n°1. Le temps d'exécution est proportionnel au nombre de bâtiments présents puisque, pour le deuxième sous-nuage, l'extraction des trois parties de bâtiments se fait approximativement en 45 secondes.

La figure ci-dessous reprend l'intégralité des bâtiments extraits en appliquant les seuils choisis. Elle regroupe les résultats obtenus pour les deux sous-nuages. Malheureusement, on y voit la présence de bruit au niveau de plusieurs portes et fenêtres. Ce bruit aura évidemment un impact sur les résultats de la segmentation/classification.

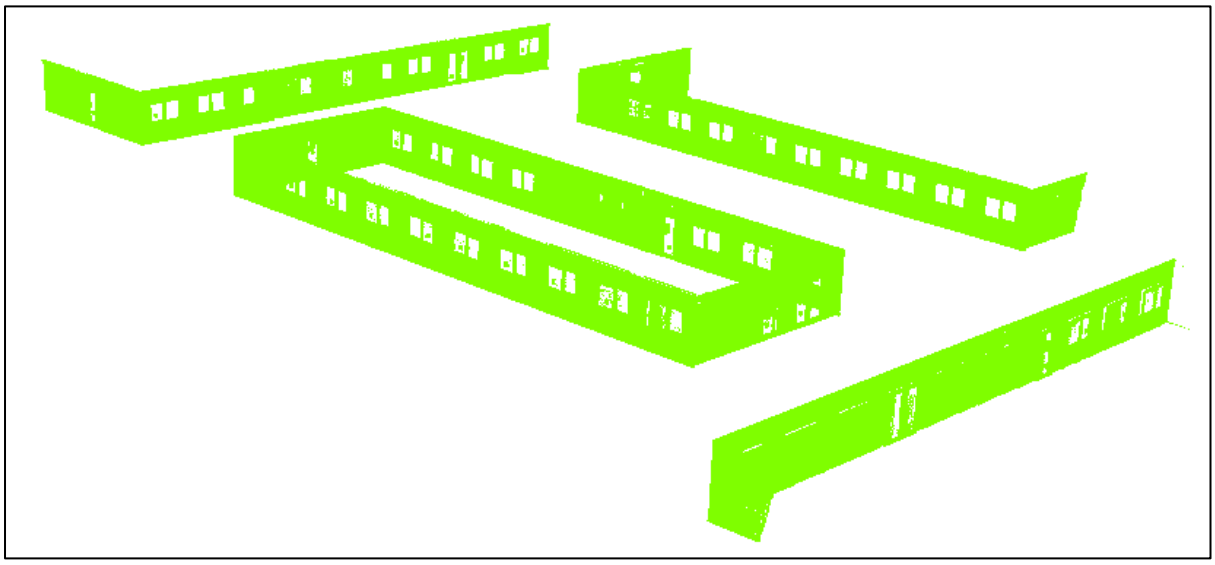


Figure 31 : Résultat de l'extraction des bâtiments par la méthode avec levé topographique

6.2.3.3. Portes et fenêtres

La segmentation des portes et des fenêtres a été faite de la même manière, en appliquant la fonction d'extraction rectangulaire (cf. 4.2.4.2.) sur les bâtiments extraits (figure 31). Tout d'abord, une marge de 2 cm a été appliquée aux coordonnées minimales et maximales (en X, Y et Z) des points délimitant l'ouverture. Cette marge de sécurité a pour objectif de limiter l'impact de l'imprécision sur les points levés afin de ne pas rejeter des points inclus dans l'objet. Ensuite, différents seuils ont été testés pour tenir compte de l'épaisseur des portes et des fenêtres. Si la distance entre le point testé et le plan correspondant à l'ouverture est inférieure au seuil, le point appartient à cette dernière.

Pour les portes, le seuil choisi est de 4 cm. La figure ci-dessous reprend les résultats obtenus pour trois seuils distincts afin de visualiser les différences. Le bruit, déjà mis en évidence lors de l'extraction des bâtiments, se retrouve dans la classe « Portes » même lors de l'application

du plus petit seuil où de nombreux points ne sont pas bien classés. Il ne semble donc pas possible de retirer ce bruit sans affecter grandement la segmentation de l'ouverture. Avec le plus grand seuil, les charnières sont mieux prises en compte mais d'autres problèmes surviennent notamment au niveau du sol où les points se retrouvent mal classés. Le seuil de 4 cm présente une situation intermédiaire qui est donc préférée.

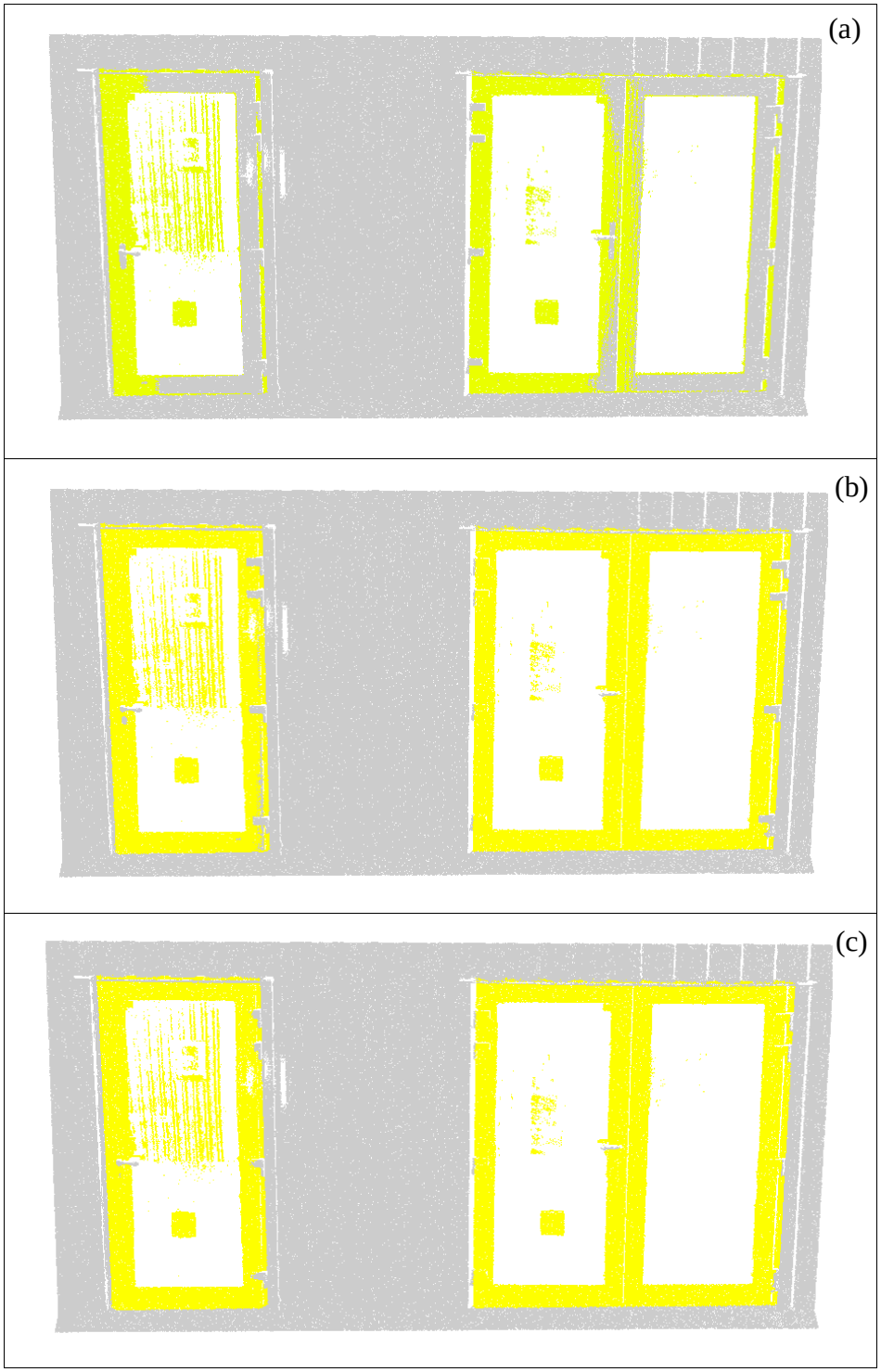


Figure 32 : Segmentation des portes par la méthode avec levé topographique
(a) Seuil = 2 cm ; (b) Seuil = 4 cm ; (c) Seuil = 6 cm

Des tests du même genre ont été effectués pour les fenêtres. La figure suivante illustre les résultats pour différents seuils. Le premier seuil (de 1 cm) permet d'éliminer le reflet de la fenêtre gauche mais il entraîne le rejet d'une partie du cadre des fenêtres. A l'inverse, le seuil de 5 cm permet une segmentation de la totalité du châssis mais il reprend également une grande partie du bruit présent. Le seuil intermédiaire de 3cm présente quant à lui un résultat satisfaisant puisque l'intégralité du cadre semble reprise dans la classe « Fenêtres » tout en éliminant le bruit de réflexion. C'est donc ce seuil qui a été choisi et appliqué à l'ensemble du nuage de points.

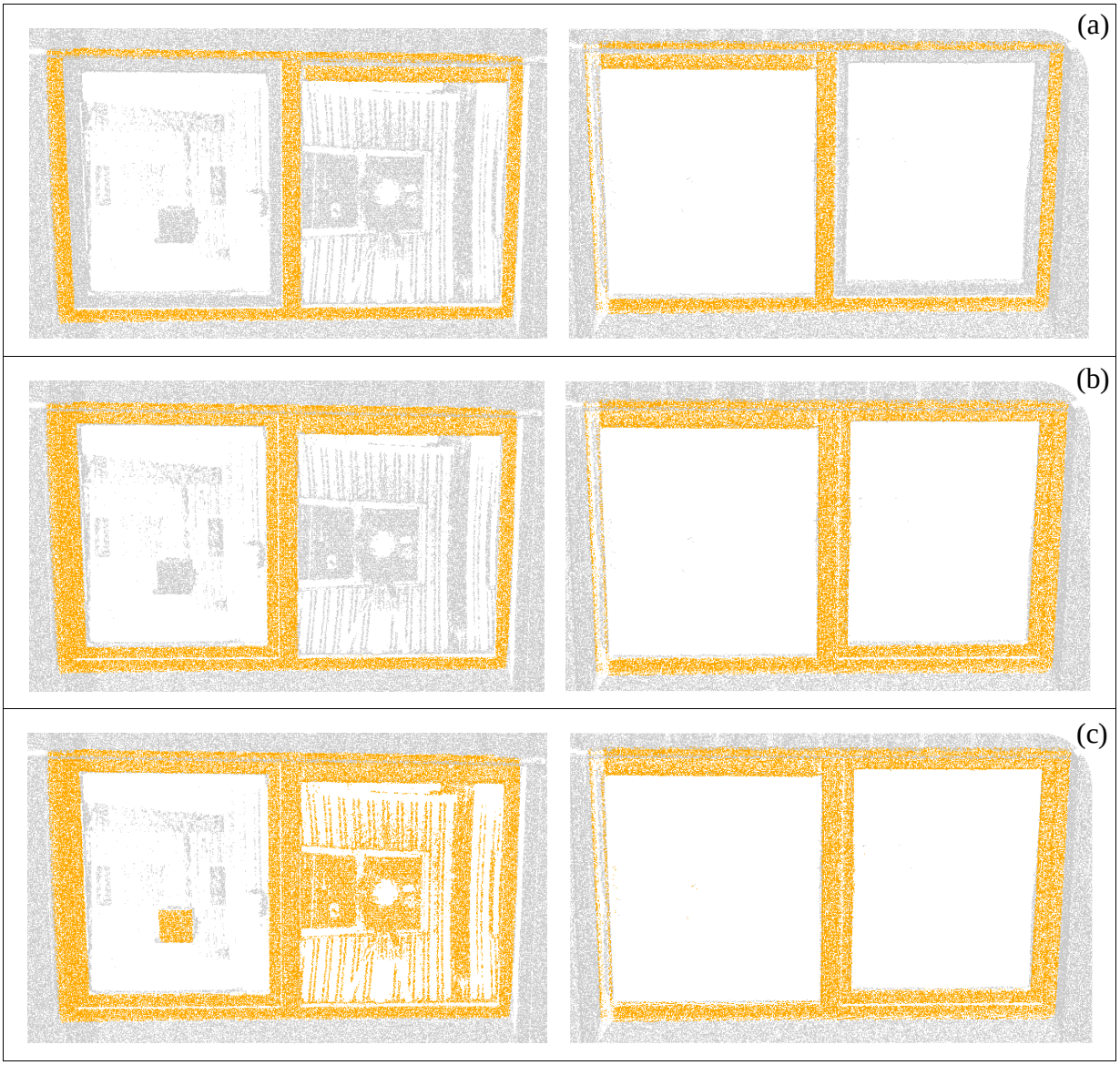


Figure 33 : Segmentation des fenêtres par la méthode avec levé topographique

(a) Seuil = 1 cm ; (b) Seuil = 3 cm ; (c) Seuil = 5 cm

Le temps d'extraction des portes est aux alentours des 10 secondes. Il est approximativement le même pour les deux sous-nuages puisqu'ils comptent le même nombre de portes (4 chacun). Les temps d'exécution sont également semblables pour les fenêtres. Ils sont toutefois plus longs étant donné qu'il y a plus de fenêtres que de portes. L'extraction se fait entre 45 et 60 secondes pour chacun des sous-nuages.

6.2.3.4. Résultat final

Le sous-nuage n°1 classifié est repris à la figure ci-dessous. A nouveau, il ne comporte que quatre classes puisque le bâtiment ne dispose pas de gouttières. Les mêmes couleurs que précédemment sont associées aux différentes classes.

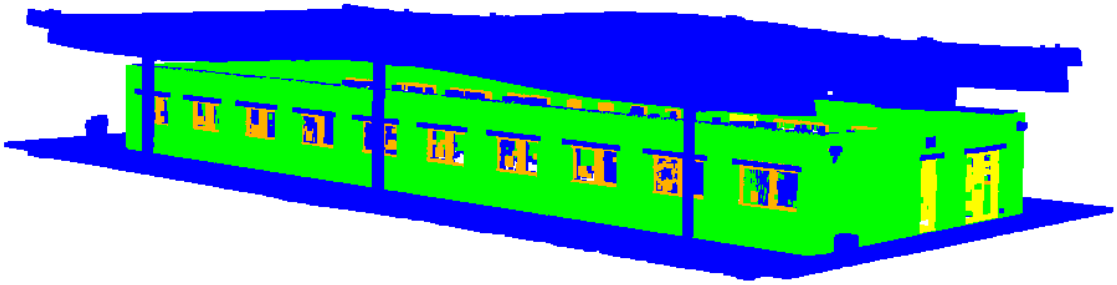


Figure 34 : Sous-nuage n°1 classifié par la méthode avec levé topographique

La figure ci-dessous présente le résultat de la classification du sous-nuage n°2. Celui-ci comprend cinq classes puisque des gouttières y sont incluses.

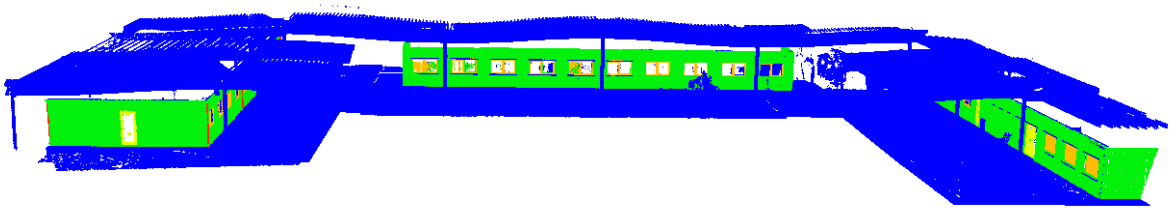


Figure 35 : Sous-nuage n°2 classifié par la méthode avec levé topographique

La figure suivante regroupe les deux sous-nuages en masquant la classe 0 contenant notamment le bruit pour mieux visualiser le résultat. La classification semble globalement satisfaisante mais plusieurs problèmes se remarquent déjà tels que la présence de bruit dans la classe « Portes » et dans la classe « Fenêtres ». Les résultats seront analysés plus en détail lors de la validation.

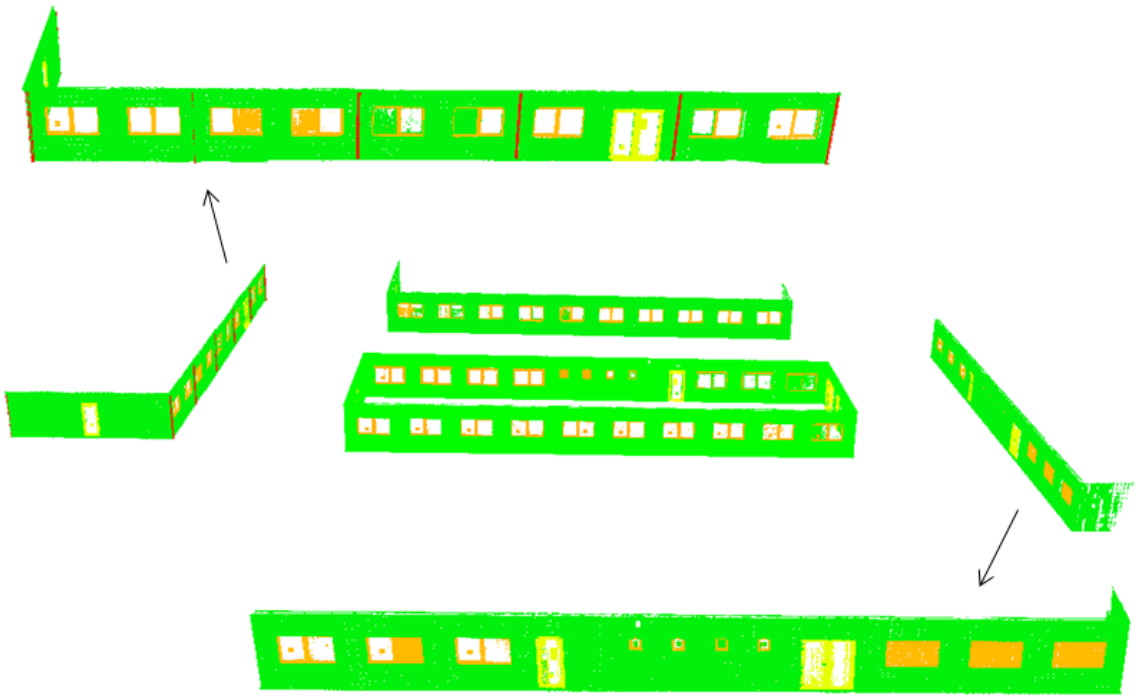


Figure 36 : Résultat final de la classification issue de la méthode avec levé topographique (en excluant la classe 0)

La durée totale de l'algorithme de segmentation/classification est d'une centaine de secondes pour le sous-nuage n°1. Pour le sous-nuage n°2, elle est d'un peu plus de 200 secondes. Contrairement à la première méthode, c'est surtout le nombre d'objets qui influence les temps de traitements et non la taille du nuage de points. La répartition des temps de traitements est présentée sur le graphique ci-dessous.

Pour cette méthode, c'est l'extraction des fenêtres qui prend le plus de temps avec pas loin de 50% de la durée totale de l'algorithme. L'écriture du fichier et l'extraction des bâtiments suivent ensuite avec un peu plus de 16% du temps total chacun.

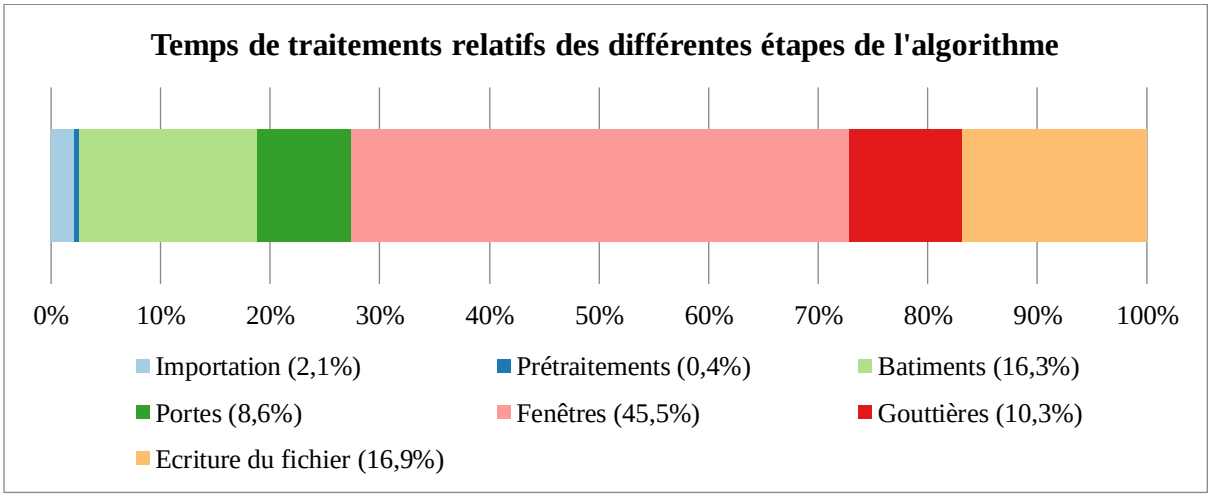


Figure 37 : Temps de traitements relatifs des étapes de l'algorithme basé sur le levé topographique

6.3. Comparaison et validation

Une fois les résultats obtenus, les deux méthodes de segmentation et de classification doivent être validées et comparées. La validation se fait sur base d'une matrice de confusion, en comparant le résultat de l'algorithme avec une segmentation manuelle, et en utilisant les différentes métriques présentées précédemment : exactitude globale, précision, rappel (*recall*) et F1-score. Ces informations permettent d'évaluer la qualité des algorithmes et de mettre en évidence les problèmes relatifs aux différentes classes. Il est toutefois à noter que la segmentation manuelle est considérée comme une vérité absolue pour la validation. Des erreurs peuvent pourtant être présentes puisque sa réalisation est subjective et fastidieuse.

Les deux méthodes sont également comparées entre elles sur base de ces résultats. Outre la qualité, les temps de traitements pour les deux méthodes et pour la segmentation manuelle sont comparés. La validation est effectuée d'abord pour le sous-nuage n°1 et ensuite pour le sous-nuage n°2.

6.3.1. Sous-nuage n°1

La matrice de confusion ci-dessous reprend les résultats de la classification pour la méthode des cibles. L'exactitude globale, calculée avec l'équation (4) sur base de cette matrice, s'élève à 97,5%. Ce résultat semble plus que satisfaisant mais il est très peu représentatif. En effet, vu la prépondérance de la classe « Reste/bruit », l'exactitude globale est biaisée et majoritairement influencée par cette classe. Pour analyser plus en détails les résultats, le tableau 5 reprend les différentes métriques associées à chaque classe ainsi que la moyenne non-pondérée de ces résultats.

Méthode des cibles		Résultat de la classification			
		Reste/bruit	Murs	Portes	Fenêtres
Vérité terrain	Reste/bruit	22 819 476	205 993	35 973	155 238
	Murs	353 555	11 351 059	22 373	29 275
	Portes	7907	512	236 413	0
	Fenêtres	72 636	0	0	496 520

Tableau 4 : Matrice de confusion du sous-nuage n°1 avec la méthode des cibles

		Reste/bruit	Murs	Portes	Fenêtres	Moyenne
Méthode des cibles	Precision	0,981	0,982	0,802	0,729	0,873
	Recall	0,983	0,966	0,966	0,872	0,947
	F1-score	0,982	0,974	0,876	0,794	0,907

Tableau 5 : Métriques obtenues pour chaque classe du sous-nuage n°1 pour la méthode des cibles

Même si la majorité des points est bien classée, plusieurs confusions entre classes sont à mettre en évidence. En terme de quantité de points, c'est entre la classe « Murs » et la classe « Reste/bruit » que la confusion semble la plus marquée. Cette confusion provient principalement de la forme particulière des murs en tôle ondulée qui s'ajustent donc difficilement à des plans. Il faut ajouter à cela la non-classification des appuis de fenêtres et autres renfoncements dans la classe « Murs ». Ceux-ci sont donc repris dans le reste alors qu'ils ne le devraient pas. Ce sont toutefois ces deux classes qui présentent les meilleurs résultats avec un F1-score de respectivement 0,982 et 0,974. La classe « Fenêtres » présente les moins bons résultats avec un F1-score de 0,794. Ceux-ci proviennent principalement d'un problème de précision, c'est-à-dire que de nombreux points sont attribués à cette classe alors qu'ils n'en font pas partie. En effet, plus de 155 000 points sont classés en fenêtres alors qu'il ne s'agit que de bruit. Ce problème a déjà été mentionné lors de la présentation des résultats. Ces points étant situés très proches de la vitre, il est difficile de les exclure de cette classe. La même difficulté survient avec les portes où la précision vaut seulement 0,8. Les résultats sont toutefois meilleurs avec un F1-score de 0,876. Le F1-score moyen fournit une évaluation globale plus représentative que l'exactitude globale pour des classes disproportionnées, il est de 0,907.

L'apport de la géocodification est bien présent puisqu'aucune confusion n'est présente entre les classes « Portes » et « Fenêtres » alors qu'il s'agit généralement d'une source de problèmes.

Pour la méthode avec levé topographique, la matrice de confusion et les métriques associées à chaque classe sont reprises dans les deux tableaux ci-dessous. L'exactitude globale est plus élevée que pour la méthode précédente puisqu'elle est de 98,9%. A nouveau, cette valeur n'est pas suffisamment représentative de la totalité des résultats comme elle est fortement influencée par la classe « Reste/bruit » et, à moindre mesure, par la classe « Murs ». Il faut donc analyser les résultats classe par classe.

Méthode par levé topographique		Résultat de la classification			
		Reste/bruit	Murs	Portes	Fenêtres
Vérité terrain	Reste/bruit	22 916 813	136 966	36 139	126 762
	Murs	4170	11 376 605	4774	70 713
	Portes	0	9306	235 526	0
	Fenêtres	7709	5375	0	556 072

Tableau 6 : Matrice de confusion du sous-nuage n°1 avec la méthode par levé topographique

		Reste/bruit	Murs	Portes	Fenêtres	Moyenne
Méthode par levé topographique	Precision	0,999	0,987	0,852	0,738	0,894
	Recall	0,987	0,993	0,962	0,977	0,980
	F1-score	0,993	0,99	0,904	0,841	0,932

Tableau 7 : Métriques obtenues pour chaque classe du sous-nuage n°1 pour la méthode avec levé

Les résultats pour cette méthode sont globalement meilleurs puisque, pour chaque classe, le F1-score présente une valeur plus élevée. Le score de la classe « Fenêtres » est amélioré de près de 5% et celui de la classe « Portes » d'un peu moins de 4% ce qui n'est pas négligeable. Le F1-score moyen est donc également plus élevé avec une valeur de 0,932. On retrouve néanmoins le même genre de soucis qu'avec la méthode précédente. En effet, avec plus de 126 000 points, le bruit se retrouve de nouveau fort présent dans la classe « Fenêtres » ce qui impacte la précision de cette classe. La même constatation est à faire pour la classe « Portes » même si la proportion est moindre. Grâce à la géocodification, les fenêtres et les portes sont toutefois à nouveau bien séparées. Par contre, par rapport à l'autre méthode, peu de points se retrouvent dans la classe « Reste/bruit » alors qu'ils ne devraient pas y être. La précision de cette classe est dès lors très proche de l'unité. Cette méthode s'illustre également par des valeurs de *recall* particulièrement élevées ce qui signifie que les éléments (murs, portes et fenêtres) sont généralement attribués à la classe à laquelle ils appartiennent vraiment. C'est la classe « Reste/bruit » qui vient s'immiscer dans les autres classes en dégradant les résultats.

Certaines situations problématiques se retrouvent donc dans les deux méthodes. C'est par exemple le cas du bruit provenant de la présence d'éléments tels que des stores derrières les fenêtres, comme ce fut déjà mentionné. La présence de fenêtres entrouvertes pose également problème pour la segmentation. Ces deux cas sont illustrés à la figure ci-dessous. Les résultats des deux algorithmes y sont comparés avec la segmentation manuelle.

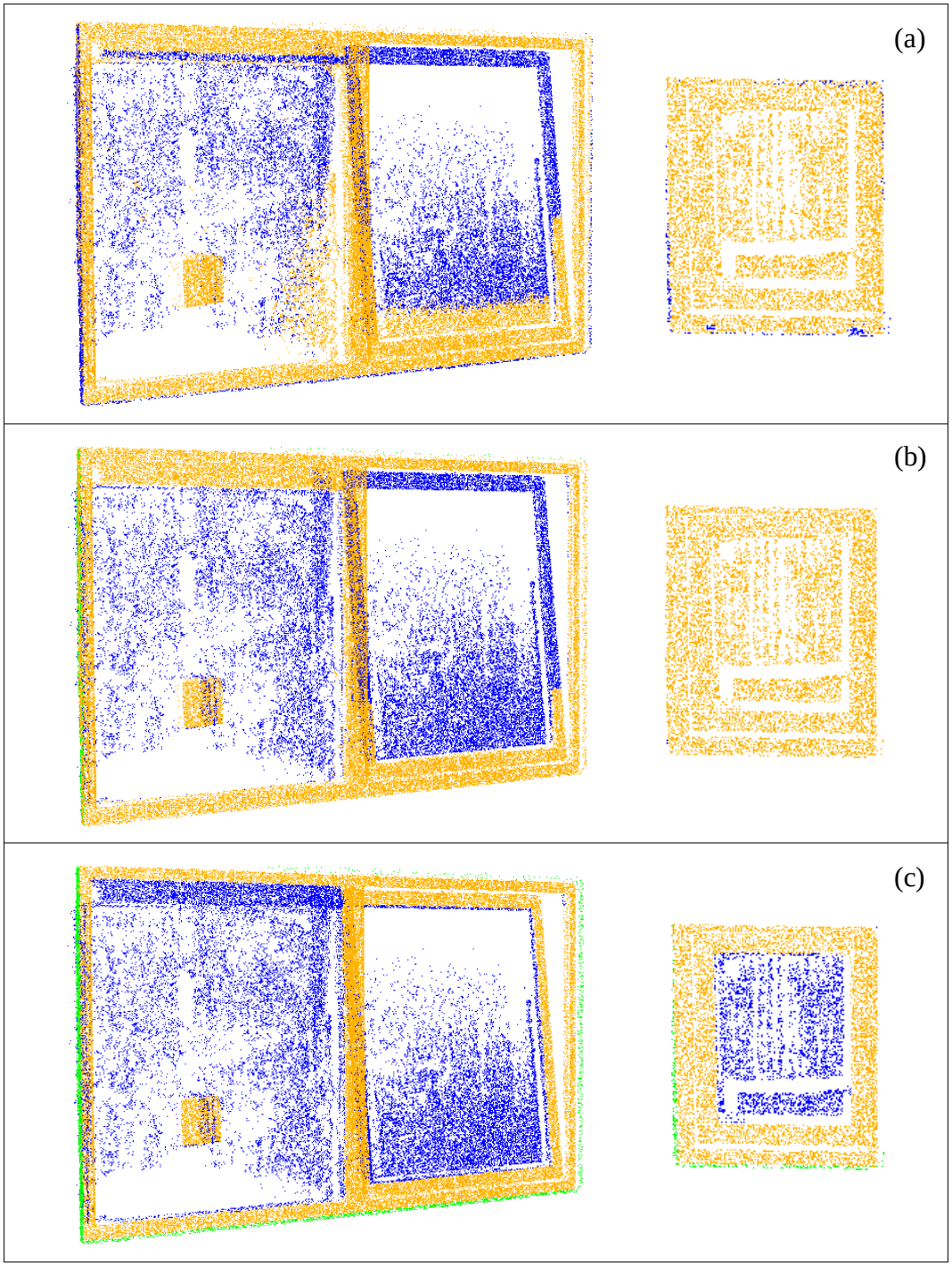


Figure 38 : Exemples de situations problématiques pour les algorithmes de segmentation (a) Méthode des cibles ; (b) Méthode par levé topographique ; (c) Segmentation manuelle

6.3.2. Sous-nuage n°2

Les matrices de confusion pour chacune des méthodes de segmentation du sous-nuage n°2 sont reprises aux deux tableaux ci-dessous. L'exactitude globale pour la méthode des cibles est de 98,4% et celle pour la méthode avec levé topographique est de 99,2%. Ces valeurs

n'étant pas représentatives, le tableau 10 reprend les résultats des différentes métriques associées à chaque classe pour chacune des deux méthodes.

Méthode des cibles		Résultat de la classification				
		Reste/bruit	Murs	Portes	Fenêtres	Gouttières
Vérité terrain	Reste/bruit	74 072 447	241 453	29 097	408 507	916
	Murs	562 294	12 811 151	51 414	51 414	3574
	Portes	7533	43 300	159 831	0	0
	Fenêtres	56 956	0	0	691 796	0
	Gouttières	4901	22 296	0	0	29 895

Tableau 8 : Matrice de confusion du sous-nuage n°2 avec la méthode des cibles

Méthode par levé topographique		Résultat de la classification				
		Reste/bruit	Murs	Portes	Fenêtres	Gouttières
Vérité terrain	Reste/bruit	74 327 659	186 663	36 927	200 829	342
	Murs	40 333	13 270 620	16 778	105 748	10 663
	Portes	0	3270	207 394	0	0
	Fenêtres	1455	45 035	0	702 262	0
	Gouttières	157	11 606	0	0	45 329

Tableau 9 : Matrice de confusion du sous-nuage n°2 avec la méthode par levé topographique

		Reste/bruit	Murs	Portes	Fenêtres	Gouttières	Moyenne
Méthode des cibles	Precision	0,991	0,977	0,781	0,601	0,869	0,844
	Recall	0,991	0,953	0,759	0,924	0,524	0,830
	F1-score	0,991	0,965	0,770	0,728	0,654	0,822
Méthode par levé topographique	Precision	0,999	0,982	0,794	0,696	0,804	0,855
	Recall	0,994	0,987	0,984	0,938	0,794	0,939
	F1-score	0,997	0,984	0,879	0,799	0,799	0,892

Tableau 10 : Métriques obtenues pour chaque classe du sous-nuage n°2 et pour chacune des méthodes

Avec un F1-score moyen de 0,892, la méthode avec levé topographique présente à nouveau des résultats de meilleure qualité que la méthode des cibles (F1-score moyen de 0,822). La différence est toutefois accentuée par rapport au sous-nuage n°1. Les différences sont surtout marquées pour la classe « Portes » et la classe « Gouttières » où le F1-score présente des valeurs supérieures de respectivement 11% et 14%. Cela s'explique notamment par les problèmes mentionnés plus tôt. En effet, le *recall* de la classe « Portes » est nettement diminué par rapport aux résultats précédents à cause des deux portes non-renfoncées d'un des bâtiments. Celles-ci sont classées en tant que murs et non en tant que portes ce qui affecte grandement la qualité des résultats étant donné qu'il n'y a que quatre portes dans ce sous-nuage. Pour les gouttières, la figure 22 met en évidence les problèmes dus à la manière d'effectuer la segmentation entrainant une confusion importante entre la classe « Gouttières » et la classe « Murs ». Il faut ajouter à cela les deux gouttières non-segmentées. La combinaison de ces deux éléments mène à un F1-score des gouttières relativement bas pour la méthode des cibles puisqu'il ne s'élève qu'à 0,654.

Par rapport au sous-nuage n°1, les résultats pour la méthode des cibles sont globalement proches, hormis pour les portes, même s'ils sont légèrement plus faibles. Le changement n'est pas significatif. La méthode s'applique donc aussi bien à des bâtiments complets qu'incomplets. Par contre, la précision pour les fenêtres est nettement plus faible pour ce nuage-ci, même si elle est compensée en grande partie par le *recall*. Cela montre l'influence du bruit et des seuils qui doivent être bien adaptés aux bâtiments rencontrés.

La méthode avec levé topographique présente également des résultats similaires à ceux obtenus pour le sous-nuage n°1. La différence majeure se trouve au niveau de la précision des fenêtres. Cette caractéristique étant commune aux deux méthodes, le problème vient probablement de la présence de bruit (stores notamment) très proche des fenêtres ce qui limite fortement son élimination. Ce phénomène se perçoit sur les deux figures présentant les résultats finaux. On y aperçoit une grande part de bruit reprise dans la classe « Fenêtres ».

6.3.3. Validation temporelle

Si, au niveau de la qualité des résultats, la méthode avec levé topographique devance légèrement l'autre, il reste à analyser les temps nécessaires à la réalisation des deux méthodes par rapport à une segmentation manuelle. Le tableau 11 reprend les temps approximatifs des

différentes étapes que sont l'acquisition, les prétraitements et la segmentation/classification. Pour cette dernière étape, les temps d'exécution pour les deux sous-nuages sont additionnés.

L'acquisition de la méthode avec levé topographique est plus longue étant donné qu'elle requiert l'acquisition de deux jeux de données. En outre, selon la disposition de la zone, le nombre de stations à faire et le nombre de points à mesurer, la durée du levé peut croître rapidement. Celle-ci est toutefois en partie contrebalancée par une segmentation/classification rapide (environ 5 minutes).

La durée d'exécution de l'algorithme de la méthode des cibles est nettement plus conséquente mais l'acquisition n'est que très peu allongée par rapport à une acquisition traditionnelle. Cette méthode est donc globalement plus rapide puisqu'elle nécessite une grosse journée de travail pour une zone de cette ampleur alors que l'autre méthode en demande deux. Il s'agit d'une différence significative.

Les deux méthodes présentées sont plus rapides qu'une segmentation manuelle, tout du moins pour un utilisateur non-expérimenté. En effet, deux journées de travail ont été consacrées à la réalisation de la segmentation manuelle. Il faut ajouter à cela l'acquisition et les prétraitements ce qui mène à environ trois jours de travail.

	Méthode des cibles	Méthode avec levé topographique	Segmentation manuelle
Acquisition	4h30	4h + 8h	4h
Prétraitements	2h	2h + 0h30	2h
Segmentation/classification	2h10	0h05	15h
Total	8h40	14h35	21h

Tableau 11 : Comparaison des temps pour les différentes méthodes

7. DISCUSSION

A partir de l'analyse des résultats, les points forts (7.1.) et les limites (7.2.) des deux méthodes développées peuvent être mis en évidence. Les perspectives découlant de ce travail sont également présentées dans ce chapitre (7.3.). Avant de détailler ces différents points, il est à noter qu'il s'agit de deux méthodes de segmentation/classification semi-automatiques. En effet, les différents paramètres (seuils, distances, etc.) sont à définir au préalable afin de correspondre au mieux aux bâtiments rencontrés. Des seuils ont été fixés dans le cadre de ce travail mais il est possible qu'ils ne soient pas les mieux adaptés à chaque situation. La nature et l'épaisseur des murs, le type de châssis ou encore la précision de l'appareil utilisé sont des éléments qui peuvent influencer l'adéquation de ces paramètres. Une fois ceux-ci fixés, l'algorithme effectue automatiquement la classification du nuage de points qui lui est fourni en entrée. Une intervention de l'utilisateur peut être nécessaire mais, comme elle est fort limitée, elle ne peut être considérée ni comme une faiblesse ni comme un point fort des méthodes.

7.1. Points forts

L'avantage principal des deux méthodes proposées vient de la possibilité d'adapter la segmentation à chaque type d'objets grâce à la géocodification. Cela permet d'utiliser une méthode de segmentation développée spécifiquement pour chaque classe et qui correspond mieux aux caractéristiques de l'objet. De plus, la segmentation/classification est effectuée uniquement pour les objets désirés et non sur la totalité du nuage de points. Le volume de données à analyser est donc réduit et les traitements sont accélérés. Ces deux éléments ne sont pas pris en compte dans les méthodes de segmentation automatique n'utilisant pas d'informations a priori où l'intégralité du nuage de points est analysée. Ils peuvent l'être lors d'une segmentation manuelle mais il n'est pas toujours aisé de repérer et distinguer les différents objets dans le nuage de points, d'autant plus si l'opérateur n'est pas familier avec la zone étudiée. Par exemple, dans ce cas-ci, les gouttières n'étaient pas réparties uniformément et leur identification dans le nuage n'était pas évidente. Le fait de les avoir mesurées ou marquées sur le terrain a permis d'assurer leur reconnaissance et leur segmentation.

Un autre point fort de ces deux méthodes est qu'elles peuvent être facilement adaptées à de nombreuses situations, aussi bien dans un environnement extérieur qu'intérieur. Outre leur

intérêt pour la segmentation du bâti, elles pourraient par exemple être utiles pour repérer et segmenter les poteaux électriques autour d'une place ou encore pour extraire les tuyaux d'aération dans un entrepôt. La segmentation est plus facilement ciblée que pour les méthodes existantes. Les confusions entre objets sont également réduites.

En outre, les résultats ont démontré que les deux méthodes permettent de gagner du temps sur une segmentation manuelle. C'est principalement le cas pour la méthode des cibles puisque elle est plus rapide que l'autre méthode.

Enfin, un dernier avantage de ces méthodes est qu'elles s'intègrent bien dans le travail du géomètre. Elles ne nécessitent pas une expertise ou des compétences supplémentaires pour pouvoir être appliquées. Elles peuvent donc aisément être mises en place pour faciliter le traitement des nuages de points.

7.2. Limites

Une des limites principales des méthodes proposées, expérimentée au niveau des gouttières notamment, est qu'un oubli ou une erreur lors de l'acquisition se répercute directement dans les résultats. En effet, si un objet n'a pas été levé ou repéré avec une cible (selon la méthode employée), il ne sera pas segmenté et classifié par l'algorithme. Il faut donc être méthodique et soigneux lors de l'acquisition pour éviter que cela n'arrive. Ce genre de problème ne se présente pas avec les méthodes de segmentation automatique existantes. C'est néanmoins déjà le cas lors de la réalisation d'un levé traditionnel, tous les éléments doivent être mesurés pour obtenir des résultats satisfaisants.

Une autre limite a été soulignée lors de l'analyse des résultats. Elle provient du fait que la méthode des cibles n'est pas applicable à toutes les structures de bâtiments. En effet, les résultats de la classification du sous-nuage n°2 montrent que plusieurs portes non-renfoncées n'ont pas été correctement segmentées et se voient attribuées à la classe « Murs ». Le même problème surviendrait avec des fenêtres non-renfoncées. L'algorithme développé pour cette méthode ne permet donc pas de distinguer les éléments situés dans les plans des murs puisque ceux-ci sont extraits en premier et ils sont ensuite mis de côté. Ce n'est pas le cas pour la méthode avec levé topographique qui est adaptée à l'extraction d'ouvertures non-renfoncées. D'autres problèmes de ce genre pourraient néanmoins survenir en appliquant les algorithmes à d'autres types de bâtiments. Cela nous amène à la limite suivante.

Celle-ci n'est pas directement liée aux méthodes présentées mais plutôt au cadre de leur développement et de leur validation. Les données utilisées pour la validation des méthodes proviennent du même nuage de points que les données utilisées pour la création des algorithmes et le choix des différents paramètres. Les deux méthodologies ne sont donc pas vraiment mises à l'épreuve lors de la validation. Il aurait fallu appliquer les deux méthodes sur une autre zone afin de tester leur efficacité dans un contexte différent.

La géocodification ayant uniquement été établie en amont de l'acquisition et de la segmentation, elle pourrait sans doute être améliorée sur base des résultats et des problèmes rencontrés. Cet élément fait partie des perspectives présentées au point suivant. Il fait néanmoins également partie des limitations puisque, suite à cela, les temps de traitement ne sont pas optimaux. C'est notamment le cas pour l'extraction des murs avec la méthode des cibles. En effet, la recherche du meilleur plan avec RANSAC est assez chronophage et n'était pas prévue au départ. Celle-ci s'est néanmoins avérée nécessaire puisque le plan extrait sur base de la cible placée sur chaque mur n'était pas suffisamment précis. Un moyen de contourner ce problème pourrait être de placer une seconde cible à l'autre extrémité du mur et de procéder de la même manière que pour les fenêtres. Cela permettrait de diminuer le temps d'extraction des murs même si plus de cibles seraient nécessaires.

Enfin, les deux méthodes présentées sont plutôt adaptées à la classification d'éléments particuliers qu'à celle d'une zone complète. L'augmentation du nombre d'objets différents à segmenter entraîne une augmentation non-négligeable du temps d'acquisition et de traitement ce qui diminue fortement l'intérêt de ces méthodes. Si l'objectif est de classer la totalité du nuage de points en de nombreuses classes, les méthodes automatiques semblent plus rapides et plus efficaces. Par contre, pour extraire des éléments spécifiques, ces deux méthodes sont intéressantes.

7.3. Perspectives d'amélioration

Plusieurs pistes pourraient être envisagées pour améliorer les deux méthodes proposées, notamment pour résoudre certaines limitations. Il faudrait par exemple tester les algorithmes avec d'autres données pour évaluer leurs performances et éventuellement améliorer la méthodologie. Il serait également intéressant de comparer les résultats de ces méthodes avec les résultats de méthodes automatiques.

Comme énoncé précédemment, la géocodification pourrait également être améliorée sur base des soucis rencontrés. Outre la possibilité d'utiliser deux cibles pour extraire un mur, la position des cibles sur les fenêtres pourrait également être modifiée. En effet, celles-ci étaient disposées de manière quelconque sur les deux ouvertures aux extrémités. Cependant, pour diminuer la marge de sécurité appliquée sur les coordonnées de ces cibles et améliorer la précision des résultats, elles pourraient être placées plus spécifiquement comme illustré à la figure 39. Les cibles actuelles y sont reprises en rouge (tout comme les fenêtres segmentées) alors que les positions recommandées sont indiquées en vert. Les conséquences engendrées par la position actuelle des cibles sont illustrées à la figure d'après (figure 40). Malgré l'utilisation d'une marge de sécurité relativement importante (1 m), certains points ne sont pas repris dans la classe « Fenêtres » puisqu'ils sont trop éloignés de l'extension délimitée par les cibles. Modifier la disposition des cibles permettrait d'appliquer une marge de sécurité plus adaptée et d'ainsi améliorer la segmentation.

Figure 39 : Amélioration du positionnement des cibles sur les fenêtres

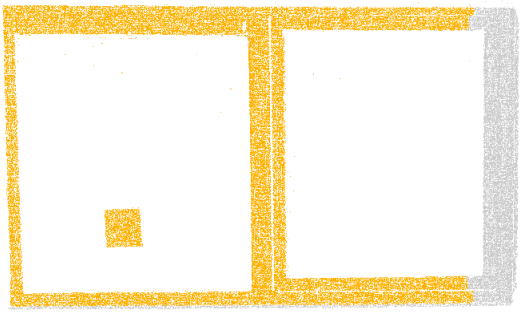


Figure 40 : Illustration du mauvais positionnement d'une cible

Une autre perspective consisterait à automatiser la détection des cibles. Au lieu de devoir associer manuellement le code correspondant à la cible, il faudrait que le logiciel puisse détecter automatiquement le numéro apposé au-dessus de la cible. La classe de l'objet serait ainsi directement connue et ne nécessiterait pas d'intervention humaine. Outre la reconnaissance des numéros, il serait possible de définir la géocodification uniquement sur base de la position et de la taille des cibles. Le voisinage pourrait aussi éventuellement être considéré pour distinguer les différentes classes. La reconnaissance automatique semble

néanmoins être l'option la plus intéressante pour ne pas être limité dans le nombre de classes et ne pas devoir se tracasser avec la taille et la disposition des cibles.

Enfin, des développements futurs pourraient être envisagés dans la continuité de ce travail. Les méthodes présentées pourraient notamment être adaptées à d'autres contextes et d'autres types d'objets. De plus, ce travail s'est consacré à l'analyse des apports d'une géocodification dans les processus de segmentation et de classification. Il serait intéressant d'étudier les apports éventuels de la géocodification au niveau de la modélisation. A titre d'exemple, grâce au positionnement de cibles, l'emplacement des vitres est connu. Elles peuvent donc être modélisées assez précisément, même si le laser passe au travers.

8. CONCLUSION

Les nuages de points 3D sont de plus en plus utilisés dans de nombreux domaines. Leur acquisition est aisée mais le traitement d'une telle quantité de données est complexe. La segmentation/classification manuelle étant chronophage et fastidieuse, de nombreuses recherches se penchent sur son automatisation.

La géocodification étant généralement utilisée par les géomètres lors de leurs levés afin de faciliter les post-traitements, ce travail cherchait à transposer ce principe au traitement des nuages de points. Deux méthodes distinctes ont été développées afin de répondre à la question suivante : « Est-il possible d'utiliser une géocodification lors de l'acquisition d'un nuage de points par laserscan pour améliorer le processus de segmentation et classification ? ». De cette question ont été tirées deux hypothèses :

1) Il est effectivement possible de créer une géocodification pour l'acquisition d'un nuage de points au laserscan et celle-ci permet d'adapter le processus de segmentation/classification aux éléments présents sur le terrain.

2) Cette méthode de segmentation/classification est plus rapide qu'une segmentation manuelle pour des résultats équivalents en précision.

Les deux méthodes implémentées pour valider ou invalider ces hypothèses se sont concentrées sur la segmentation du bâti avec l'objectif de distinguer les murs, portes, fenêtres et éventuelles gouttières du reste du nuage de points.

La première des méthodes proposées se base sur l'utilisation de cibles auxquelles un code spécifique est associé. Celles-ci sont ensuite disposées sur les éléments d'intérêts afin de guider le processus de segmentation. Avec un F1-score moyen de 0,90 pour le sous-nuage n°1 et 0,82 pour le n°2, les résultats obtenus pour cette méthode sont plutôt satisfaisants même si quelques problèmes persistent. Une grosse journée de travail est nécessaire pour effectuer l'intégralité du processus, de l'acquisition à la classification finale.

La seconde méthode intègre quant à elle des données issues d'un levé topographique géocodifié dans le processus de segmentation. La qualité des résultats pour cette méthode est supérieure à la méthode précédente puisque le F1-score moyen obtenu pour le sous-nuage n°1

est de 0,93 et celui pour le second sous-nuage est de 0,89. Cette méthode nécessite toutefois une journée de travail supplémentaire consacrée à l'acquisition des données.

Ces deux méthodes permettent dès lors de vérifier la première hypothèse émise puisque deux géocodifications différentes ont été créées et utilisées pour adapter le processus de segmentation/classification. Par contre, seule la première partie de la seconde hypothèse est vérifiée. En effet, les deux méthodes développées sont plus rapides qu'une segmentation manuelle mais, avec un F1-score moyen compris entre 0,82 et 0,93, la qualité des résultats obtenus se rapproche de celle d'une segmentation manuelle mais ne l'égale pas.

9. BIBLIOGRAPHIE

Anaconda Navigator (version 1.10.0) [Computer software]. (2020). https://anaconda.com. Consulté le 23 avril 2021.

Billen, R. (2018). Mesures d'angles (cours 3b). *Topographie*. Notes de cours de bac en sciences géographiques, Liège, Université de Liège, inédit, 78 p.

Biosca, J.M. & Lerma, J.L. (2008). Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. *ISPRS Journal of Photogrammetry and Remote Sensing*, 63(1), 84-98. https://doi.org/10.1016/j.isprsjprs.2007.07.010.

Brown, G. & Butler, H. (2014). *Laspy: Python library for lidar LAS/LAZ IO*. Laspy. https://laspy.readthedocs.io/en/latest/. Consulté le 22 mars 2021.

Cabo, C., Garcia Cortés, S. & Ordoñez, C. (2015). Mobile Laser Scanner data for automatic surface detection based on line arrangement. *Automation in Construction*, 58, 28-37. https://doi.org/10.1016/j.autcon.2015.07.005.

Che, E. & Olsen, M.J. (2018). Multi-scan segmentation of terrestrial laser scanning data based on normal variation analysis. *ISPRS Journal of Photogrammetry and Remote Sensing*, 143, 233-248. https://doi.org/10.1016/j.isprsjprs.2018.01.019.

Che, E., Jung, J. & Olsen, M.J. (2019). Object recognition, segmentation, and classification of mobile laser scanning point clouds: A state of the art review. *Sensors*, 19(4). https://doi.org/10.3390/s19040810.

CloudCompare (version 2.10.2) [GPL software]. (2021). http://www.cloudcompare.org/. Consulté le 12 avril 2021.

Djemâ, N. (2018). *Segmentation d'un nuage de points obtenu par scanner laser sur base d'un levé topographique classique*. Mémoire de master en sciences géographiques, orientation géomatique et géométrologie, Liège, Université de Liège, inédit, 64 p. http://hdl.handle.net/2268.2/4317.

Dong, Z., Yang, B., Hu, P. & Scherer, S. (2018). An efficient global energy optimization approach for robust 3D plane segmentation of point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 137, 112-133. https://doi.org/10.1016/j.isprsjprs.2018.01.013.

Fischler, M.A. & Bolles, R.C. (1981). Random Sample Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 381-395.

Grilli, E., Menna, F. & Remondino, F. (2017). A review of point clouds segmentation and classification algorithms. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII(2), 339-344. https://doi.org/10.5194/isprs-archives-XLII-2-W3-339-2017.

Han, X.-F., Jin, J.S., Wang, M.-J., Jiang, W., Gao, L. & Xiao, L. (2017). A review of algorithms for filtering the 3D point cloud. *Signal Processing : Image Communication*, 57, 103-112. https://doi.org/10.1016/j.image.2017.05.009.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T.E. (2020). Array programming with NumPy. *Nature*, 585, 357-362. https://doi.org/10.1038/s41586-020-2649-2.

Hough, P.V. (1962). *Method and means for recognizing complex patterns.* United States.

Hunter, J.D. (2007). Matplotlib : A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55.

Jakovljevic, G., Govedarica, M., Alvarez-Taboada, F. & Pajic, V. (2019). Accuracy assessment of deep learning based classification of LiDAR and UAV points clouds for DTM creation and flood risk mapping. *Geosciences*, 9(7), 323. https://doi.org/10.3390/geosciences9070323.

Jonlet, B. & Poux, F. (2017). La géocodification en général/Quelques possibilités de Covadis. *Projet intégré de "Géomètre-expert"*. Notes de cours de master en sciences géographiques, orientation géomatique, à finalité géomètre-expert, Liège, Université de Liège, inédit, 41 p.

Landes, T. & Grussenmeyer, P. (2011). Les principes fondamentaux de la lasergrammétrie terrestre : systèmes et caractéristiques (partie 1/2). *XYZ*, 128, 37-49.

Landes, T., Grussenmeyer, P. & Boulaassal, H. (2011). Les principes fondamentaux de la lasergrammétrie terrestre : acquisition , traitement des données et applications (partie 2/2). *XYZ*, 129, 25-38.

Leica Geosystems. (2016). *Leica ScanStation P40/P30 : System Field Manual.* HeerBrug, Suisse: Leica Geosystems AG.

Lesecque, F. (2019). *Extraction et utilisation de connaissances métier structurées pour la classification de nuages de points 3D.* Mémoire de master en sciences géographiques, orientation géomatique et géométrologie, Liège, Université de Liège, inédit, 120 p. http://hdl.handle.net/2268.2/7363.

Nguyen, A. & Le, B. (2013). 3D point cloud segmentation : A survey. *In 2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, November 12-15, 2013, Manila, Philippines. IEEE, 225-230. https://doi.org/10.1109/RAM.2013.6758588.

Nurunnabi, A., Belton, D. & West, G. (2016). Robust segmentation for large volumes of laser scanning three-dimensional point cloud data. *IEEE Transactions on Geoscience and Remote Sensing*, 54(8), 4790-4805. https://doi.org/10.1109/TGRS.2016.2551546.

Oude Elberink, S.J. (2020). Smart fusion of mobile laser scanner data with large scale topographic maps. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-2-2020, 251-258. https://doi.org/10.5194/isprs-annals-V-2-2020-251-2020.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn : Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.

Poux, F. & Billen, R. (2019). Voxel-based 3D point cloud semantic segmentation: Unsupervised geometric and relationship featuring vs Deep Learning methods. *ISPRS International Journal of Geo-Information*, 8(5), 213. https://doi.org/10.3390/ijgi8050213.

Poux, F., Hallot, P., Jonlet, B., Carré, C. & Billen, R. (2014). Segmentation semi-automatique pour le traitement de données 3D denses : application au patrimoine architectural. *XYZ,* 141(4), 69-75.

Poux, F., Hallot, P., Neuville, R. & Billen, R. (2016). Smart point cloud: definition and remaining challenges. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences,* IV(2), 119-127. https://doi.org/10.5194/isprs-annals-IV-2-W1-119-2016.

Qi, C.R., Su, H., Mo, K. & Guibas, L.J. (2017). PointNet : Deep learning on point sets for 3D classification and segmentation. *In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Honolulu, USA, 21–26 July 2017. IEEE, 77–85. https://doi.org/10.1109/CVPR.2017.16.

Schnabel, R., Wahl, R. & Klein, R. (2007). Efficient RANSAC for point-cloud shape detection. *Computer Graphics Forum,* 26(2), 214-226. https://doi.org/10.1111/j.1467-8659.2007.01016.x.

Theiler, P.W., Wegner, J.D. & Schindler, K. (2015). *Automatic registration of partially overlapping terrestrial laser scanner point clouds.* ETH Zürich. https://prs.igp.ethz.ch/research/completed_projects/automatic_registration_of_point_clouds.html. Consulté le 20 juillet 2021.

Van Genechten, B. (2008). *Theory and practice on Terrestrial Laser Scanning : training material based on practical application* (éd. 4). Valencia: Universidad Politecnica de Valencia Editorial, 261 p.

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., … & Scipy 1.0 Contributors. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17, 261-272. https://doi.org/10.1038/s41592-019-0686-2.

Vo, A.V., Truong-Hong, L., Laefer, D.F. & Bertolotto, M. (2015). Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104, 88-100. https://doi.org/10.1016/j.isprsjprs.2015.01.011.

Vosselman, G., Gorte, B.G.H., Sithole, G. & Rabbani, T. (2004). Recognising structure in laser scanning point clouds. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences,* XXXVI(8), 33-38.

Xie, Y., Tian, J. & Zhu, X.X. (2020). Linking points with labels in 3D: a review of point cloud semantic segmentation. *IEEE Geoscience and Remote Sensing Magazine*, 8(4), 38-59. https://doi.org/10.1109/MGRS.2019.2937630.

Yang, B., Zang, Y., Dong, Z. & Huang, R. (2015). An automated method to register airborne and terrestrial laser scanning point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 109, 62-76. https://doi.org/10.1016/j.isprsjprs.2015.08.006.

Zolanvari, S.M.I & Laefer, D.F. (2016). Slicing Method for curved façade and window extraction from point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing,* 119, 334-346. https://doi.org/10.1016/j.isprsjprs.2016.06.011.

Zolanvari, S.M.I, Laefer, D.F. & Natanzi, A.S. (2018). Three-dimensional building façade segmentation and opening area detection from point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing,* 143, 134-149. https://doi.org/10.1016/j.isprsjprs.2018.04.004.

10. ANNEXES

10.1. Algorithme de segmentation pour la méthode des cibles

```
#------------------------------------------------------------------------------#
#
#                Script de segmentation / classification
#                 sur base du positionnement de cibles
#
#------------------------------------------------------------------------------#

#  Importation des librairies et des fonctions
#**********************************************

import numpy as np
import time
import laspy as lp
from math import sqrt
from scipy.spatial.distance import cdist
from math import pi, ceil
import random

#------------------------------------------------------------------------------#
#
#                              Fonctions
#                             -----------

# Détermination du voisinage des points
#--------------------------------------

# Calcul du voisinage sphérique "brut"
def brute_force_spherical(queries, supports, radius, coord = True):
# Cette fonction va parcourir les points et les définir en tant que voisin d'un point
↪  (query) si la distance entre les 2 points est plus petite qu'un seuil (radius)

    neighbors = {}     #Dictionnaire qui contiendra les voisins de chaque point
    i = 0

    for requete in queries:    #Boucle sur les pts dont on cherche les voisins
        dist = {}              #Dictionnaire qui contiendra toutes les distances entre
        ↪  les points et la query
        dist[i] = np.linalg.norm(requete-supports, axis=1)                #Distance
        ↪  entre la requête et tous les autres points
        neighbors[i] = np.where((dist[i]<radius) & (dist[i] != 0))[0]      #On garde
        ↪  les voisins si distance inférieure au seuil

        if coord == True:
            voisins = []
            for value in neighbors[i]:          #Pour chaque voisin d'un point, on va
            ↪  remplace l'ID du point par ses coordonnées
                voisins.append(supports[value])
            neighbors[i] = voisins              #Le dictionnaire contiendra pour chaque
            ↪  point les coordonnées des points voisins

        i+=1

    return neighbors

# Calcul du voisinage sphérique en utilisant une grille de voxels pour ne pas calculer
↪  le voisinage sur tous le nuage mais seulement sur les points proches
class neighborhood_grid():
    def __init__(self, points, voxel_size):    #Création de la grille de voxels

        self.grid_voxel_size = voxel_size
        self.points = points

        min_x = np.min(points[:,0])     #Calcul du minimum en x
        min_y = np.min(points[:,1])     #Minimum en y
        min_z = np.min(points[:,2])     #Minimum en z

        dico={}
        pts = 0
        while pts < len(points):        #Parcours de tous les points
            indice_x = (points[pts,0]-min_x)//voxel_size    #Indice en x pour chaque pt
            ↪  en utilisant le min en x et la taille des voxels
            indice_y = (points[pts,1]-min_y)//voxel_size    #Indice en y
            indice_z = (points[pts,2]-min_z)//voxel_size    #Indice en z
```

```
            tuple_indice = (int(indice_x), int(indice_y), int(indice_z))       #Indice
            ↪ des voxels converti sous forme de tuple

            if tuple_indice not in dico:         #Insertion de l'indice dans le dico si
            ↪ il n'y est pas encore
                dico[tuple_indice] = [pts]       #Coordonnées en x, y et z du point
                ↪ insérées avec la clé du voxel correspondant

            else:
                dico[tuple_indice].append(pts)   #Si indice déjà présent, ajout dans la
                ↪ liste des coordonnées pour chaque point du même voxel

            pts+=1

        self.dico = dico
        self.min_x = min_x
        self.min_y = min_y
        self.min_z = min_z

    def query_radius(self, queries, radius, coord = True):

        neighbors,neighborhoods = {}, {}

        q = 0                    #Incrémentation des queries en démarrant à 0
        while q < len(queries):   #Calcul du voisinage des différentes requêtes

            voxel = (int((queries[q][0] - self.min_x)/self.grid_voxel_size),
            ↪ int((queries[q][1] - self.min_y)/self.grid_voxel_size),
            ↪ int((queries[q][2] - self.min_z)/self.grid_voxel_size))  #
            ↪ Détermination du voxel dans lequel se trouve le point dont on cherche
            ↪ le voisinage
            larg = ceil(radius/self.grid_voxel_size)          # Calcul du nombre entier
            ↪ de voxels à gauche (et à droite) du voxel où se trouve la query

            # Parcours des voxels dans le voisinage/rayon de la requête à l'aide de 3
            ↪ boucles (en x, en y et en z)
            key = (queries[q][0], queries[q][1], queries[q][2])
            x = voxel[0] - larg              #Initialisation de l'indice en x du voxel
            ↪ à partir duquel on va chercher les voisins
            while x <= voxel[0] + larg:      #Boucle parcourant l'indice en x

                y = voxel[1] - larg          #Initialisation de l'indice en y
                while y <= voxel[1] + larg:

                    z = voxel[2] - larg      #Initialisation de l'indice en z
                    while z <= voxel[2] + larg:

                        if self.dico.get((x, y, z)) != None:        #Si la clé n'existe
                        ↪ pas, cela signifie que l'on se trouve en dehors de la grille

                            if key not in neighbors:                #Insertion de
                            ↪ l'indice dans le dico si il n'y est pas
                                neighbors[key] = self.dico[x,y,z]   #Association des
                                ↪ coord du point à la clé/query
                            else:
                                neighbors[key]=
                                ↪ np.concatenate((neighbors[key],self.dico[x,y,z]))
                                ↪   #Si la clé existe déjà, ajout des coordonnées du
                                ↪ point dans la liste
                        z+=1
                    y+=1
                x+=1

            supports = self.points[neighbors[key]]        #Création d'un nuage de
            ↪ points plus petit contenant les points des voxels autour de la query

            # Récupération des coordonnées des voisins
            if coord == True:
                voisins_radius = brute_force_spherical([queries[q]],supports,radius)
                ↪    #Calcul du voisinage sphérique sur ce sous-nuage
                neighborhoods[q] = voisins_radius[0]

            # Récupération des indices des voisins
            elif coord == False:
                voisins_radius = brute_force_spherical([queries[q]],supports,radius,
                ↪ coord = False)

                # On a les indices des voisins dans le sous-nuage mais il faut
                ↪ récupérer les indices correspondants dans le nuage entier
                if len(voisins_radius[0]) > 0:
                    neighborhoods[q] = neighbors[key][voisins_radius[0]]
```

```
                else:
                    neighborhoods[q] = []
            q+=1

        return neighborhoods

#-------------------------------------------------------------------------------

# Analyse en composantes principales afin d'obtenir les valeurs et vecteurs propres
↪ d'un ensemble de données/points
def PCA(data, sort = True):

    centroide = np.mean(data, axis=0)       #Calcul du centroïde du voisinage
    data_ajust = data - centroide           #Ajustement des données pour pouvoir créer
    ↪ la matrice de covariance: data - centroïde
    matrice_cov = np.cov(data_ajust.T)      #Calcul de la matrice de covariance

    eigenvalues, eigenvectors = np.linalg.eig(matrice_cov)  #Valeurs et vecteurs propres

    if sort:
        # Tri des valeurs et vecteurs propres (du plus grand au plus petit)
        sort = eigenvalues.argsort()[::-1]
        eigenvalues = eigenvalues[sort]
        eigenvectors = eigenvectors[:,sort]

    return eigenvalues, eigenvectors
#-------------------------------------------------------------------------------

# Fonction permettant de dèterminer le plan tangent à un point sur base de son
↪ voisinage (équation avec 4 paramètres : a, b, c et d)
def best_fitting_plane(points, param = True):

    val_prop, vect_prop = PCA(points)  #Valeurs propres et vecteurs propres

    v0 = vect_prop[:,2]                #Vecteur normal au plan tangent au point
    point = np.mean(points, axis=0)

    if param == True:
        # Détermination des paramètres du plan
        a = v0[0]           #1e composante du vecteur normal
        b = v0[1]           #2e composante du vecteur normal
        c = v0[2]           #3e composante du vecteur normal
        d = -(a*point[0] + b*point[1] + c*point[2])   #Calcul de d sur base de
        ↪ l'équation d'un plan: ax + by + cz + d = 0 (en utilisant les coordonnées du
        ↪ point dans le plan)
        return a, b, c, d

    else :
        return v0

#-------------------------------------------------------------------------------

#Fonction d'extraction des points appartenant à un plan (distance inférieure au seuil
↪ fixé) à partir de son équation (paramètres a,b,c et d).
def plane(nuage, a, b, c, d, seuil, unique = False):

    boolean = np.zeros(len(nuage), dtype = bool)

    # Pour chaque point du nuage, calcul de la distance entre le point et le plan pour
    ↪ vérifier son appartenance au plan
    i = 0
    while i < len(nuage):
        dist = abs(a*nuage[i][0] + b*nuage[i][1] + c*nuage[i][2] + d)/(sqrt( a*a + b*b
        ↪ + c*c))
        if dist < seuil:
            boolean[i] = True    #Si distance inférieure au seuil, renvoie True
        i += 1

    #Fonction retournant les points appartenant au plan et le reste du nuage
    if unique:
        segment = nuage[boolean]        #Points du nuage appartenant au segment
        remains = nuage[~boolean]       #Points restants
        return segment, remains

    #Fonction retournant un tableau booléen distinguant les points faisant partie du
    ↪ segment/plan ou non
    else:
        return boolean
```

```
#-------------------------------------------------------------------------------
↪

# Fonction calculant la distance entre un point et tous les autres pour garder ceux
↪  dont la distance est inférieure à un seuil donné
def dist(point, nuage, seuil):
    dist = np.linalg.norm(point-nuage, axis=1)       #Fonction calculant la distance
    ↪  euclidienne entre la requête et tous les autres points
    boolean = dist < seuil                           #Tableau renvoyant True si la
    ↪  distance entre la requête et le point testé est inférieure au seuil
    return boolean

#-------------------------------------------------------------------------------

# Fonction calculant les normales des points sur base de leur voisinage
def compute_normals(points, radius):

    normals = points * 0

    #Détermination du voisinage de tous les points en utilisant la méthode des voxels
    ↪  pour calculer le voisinage sphérique
    grid = neighborhood_grid(points, radius)
    voisinage = grid.query_radius(points, radius)

    #Parcours du dico avec les voisins de chaque pt pour calculer leur normale
    pts = 0
    for key in voisinage:
        voisins = voisinage[pts]

        #3 voisins minimum pour faire l'ACP et déterminer la normale de point
        if len(voisins) >= 3:
            val_prop, vect_prop = PCA(voisins)     #Valeurs et vecteurs propres
            normals[pts] = vect_prop[:,2]          #Normale du point

        #Si pas assez de voisins, normale considérée égale à 0
        else:
            normals[pts] = 0
        pts += 1

    return normals

#-------------------------------------------------------------------------------

# Fonction déterminant si deux points appartiennent à une même région (un même objet)
↪  sur base de leurs normales
def region_criterion(n1, n2):

    #Definition du seuil d'angle entre les normales pour déterminer si 2 points voisins
    ↪  appartiennent au même objet
    seuil_ang = pi/10

    #Calcul de l'angle entre les normales
    angle=np.arccos(np.clip(np.dot(n1,n2)/np.linalg.norm(n1)/np.linalg.norm(n2),-1,1))

    #Egalement condition de pseudo-horizontalité de la normale
    #Si les critères sont remplis, les 2 points font partie de la même région
    if (((abs(angle) < seuil_ang) or (abs(angle - pi) < seuil_ang)) and (n2[2] < 0.1)):
        return True
    else:
        return False

# -------------------------------------------------------------------------------

# Fonction déterminanat les points appartenant à une même region en utilisant le
↪  critère précédent
def RegionGrowing(cible, cloud, normals, radius):

    #Initialisation de la région de la taille de cloud et remplie de 0 (False)
    N = len(cloud)
    region = np.zeros(N, dtype=bool)

    #Création de la grille de vozels qui servira pour calculer le voisinage
    grid = neighborhood_grid(cloud, radius)

    #Choix d'une graine, correspondant au point du nuage le plus proche de la cible,
    ↪  qui est mise dans la file d'attente
    graine = cdist([cible], cloud).argmin()
    file =[graine]
    region[graine] = True       #Cette graine est le premier point de la région
```

```
    #Parcours de la file tant qu'il y a des points dedans
    j = 0
    while j < len(file):
        #Calcul du voisinage de la graine/point dans la file d'attente (on récupère les
        ↪  indices des voisins et non leurs coordonnées, coord = False)
        voisins = grid.query_radius([cloud[file[j],:]], radius, coord = False)

        #Pour chaque voisin qui n'est pas encore dans la région, on teste le critère
        ↪  d'appartenance à la même région
        for value in voisins[0]:
            if region[value] == False:
                region[value] = region_criterion(normals[file[j],:], normals[value])

                #Si le critère est respecté, ajout du point dans la file d'attente et
                ↪  il appartient à la région -> True
                if region[value] == True:
                    file.append(value)
        j += 1
    return region

# ----------------------------------------------------------------------------

# Fonction créant une grille 2D (système x-y) de pixels de dimension fixée (cell-size)
↪  et renvoyant les indices des points de chaque pixel
def grid_2d(points, cell_size):

    min_x = np.min(points[:,0])     #Calcul du minimum en x
    min_y = np.min(points[:,1])     #Calcul du minimum en y
    dico={}                         #Création d'un dico vide qui contiendra les indices
    ↪  de chaque pixel associés aux points se trouvant dans ces pixels

    pts = 0
    while pts < len(points):        #Parcours de tous les points
        indice_x = (points[pts,0]-min_x)//cell_size        #Indice en x pour chaque
        ↪  point en utilisant le min en x et la taille des pixels
        indice_y = (points[pts,1]-min_y)//cell_size        #Indice en y
        tuple_indice = (int(indice_x), int(indice_y))

        if tuple_indice not in dico:     #Insertion de l'indice dans le dico si il n'y
        ↪  est pas
            dico[tuple_indice] = [pts]   #Coordonnées du point insérées
        else:
            dico[tuple_indice].append(pts)   #Si indice déjà présent, ajout dans la
            ↪  liste des coordonnées pour chaque point du même pixel

        pts+=1
    return dico

# ----------------------------------------------------------------------------

# Fonction écrivant les résultats (coord x, y, z + classe) dans un fichier .las
def laswrite(nuage, filename, header):
    output = lp.file.File(filename, mode = "w", header = header)
    if(len(nuage)>0):
        output.x = nuage[:,0]
        output.y = nuage[:,1]
        output.z = nuage[:,2]
        output.raw_classification = nuage[:,3]
        print("Fichier .las enregistré !")
    else:
        print("Pas de points dans le nuage !")
    output.close()

#-----------------------------------------------------------------------------
#
#                                 Main
#                               --------
if __name__ == '__main__':

    #    Importation des données
    ##############################

    #Nuage de points au format .las
    t0 = time.time()
    point_cloud = lp.file.File('../Data/Scan/nuage_bat1_memoire.las')
    nuage = np.vstack((point_cloud.x, point_cloud.y, point_cloud.z)).T

    #Cibles (ID, X, Y, Z, code) au format .txt
    cibles = np.loadtxt('../Data/Scan/Scan_cibles.txt', delimiter=';')
    t1 = time.time()
    print('Importation faite en {:.3f} secondes'.format(t1 - t0))
```

```
        #  Segmentation et classification du nuage de points
        ##################################################

        #Séparation des cibles selon leur code pour les regrouper par catégories
        cible_bat = []        #Liste vide qui contiendra les cibles de bâtiments
        cible_door = []       #Portes
        cible_window = []     #Fenêtres
        cible_gout = []       #Gouttières
        cible_ref = []        #Cibles uniquement utiles à l'assemblage/référencement
        for i in cibles:
            if i[4] == 1:
                cible_bat.append(i[1:4])
            elif i[4] == 2:
                cible_door.append(i[1:4])
            elif i[4] == 3:
                cible_window.append(i[1:4])
            elif i[4] == 4:
                cible_gout.append(i[1:4])
            elif i[4] == 99:
                cible_ref.append(i[1:4])
            else:
                print("Code inconnu !")

        cible_bat = np.asarray(cible_bat)
        cible_door = np.asarray(cible_door)
        cible_window = np.asarray(cible_window)

    cible_gout = np.asarray(cible_gout)
    cible_stat = np.asarray(cible_ref)
    t2 = time.time()
    print('Cibles séparées en {:.3f} secondes'.format(t2 - t1))

    classif = np.zeros(len(nuage))                    #Création d'une colonne de 0 qui
    ↪  définira la classe de chaque point
    nuage = np.column_stack((nuage, classif))         #Ajout de cette colonne

    t3 = time.time()
    radius = 0.05           #Rayon de recherche du voisinage

#                           Bâtiments
#-------------------------------------------------------------------------------

    max_dist = 0.06       #Distance au plan max pour retenir les points
    voisinage = brute_force_spherical(cible_bat, nuage[:,0:3], radius, coord = True)
    ↪   #Voisinage des cibles des murs

    k = 0
    j = 0
    for key in voisinage:
        voisins = voisinage[k]                  #Voisinage de la cible traitée

        if len(voisins) >= 3:
            n1 = best_fitting_plane(voisins, False)         #Normale du plan
            a,b,c,d = best_fitting_plane(voisins)           #Paramètres du plan
            sous_nuage, exclus = plane(nuage, a, b, c, d, 1, True)    #On ne garde que
            ↪  les points situés à 1 m de ce plan comme candidats au plan final

            # RANSAC
            i = 0
            best_points = 0
            max_iter = 30               #Nombres d'itérations
            while i < max_iter:
                nbr_points = 0

                ind_sample = random.sample(range(len(sous_nuage)),2)
                ↪              #On prend 2 points aléatoires dans le sous-nauge
                sample = np.concatenate(([cible_bat[k,:]],sous_nuage[ind_sample,0:3]),
                ↪  axis = 0)    #Les 3 points déterminant le plan testé sont la cible
                ↪  et les deux points aléatoires
                n_sample = best_fitting_plane(sample, False)
                ↪              #Normale du plan testé

                angle = np.arccos(np.clip(np.dot(n1,n_sample)/np.linalg.norm(n1)/np.lin
                ↪  alg.norm(n_sample), -1, 1))  #Angle entre les
                ↪  normales

                if angle < (pi/20):        #On ne teste le plan que si sa normale est
                ↪  proche de la normale initiale (n1)
                    a,b,c,d = best_fitting_plane(sample)      #Détermination des
                    ↪  paramètres du plan testé
                    in_plane = plane(sous_nuage, a, b, c, d, max_dist, unique = False)
                    ↪  #Attribution des points si situés à une distance inf à max_dist
```

```
                nbr_points = np.count_nonzero(in_plane)   #On compte le nombre de
                ↪ points dans le plan

                if nbr_points > best_points:     #Si plan meilleur, on le garde
                    best_points = nbr_points
                    best_plane = in_plane

                i +=1

            if j == 0:       #Si 1e plan, on sort les points du mur
                murs = sous_nuage[best_plane]
                j += 1
            else :           #Sinon, on joint les points à ceux des autres murs
                murs = np.concatenate((murs,sous_nuage[best_plane]))

            nuage = np.concatenate((exclus,sous_nuage[~best_plane]))    #On regroupe
            ↪ les points qui ne font pas partie des murs

        k += 1

    #Pour éliminer les points au-dessus du bâtiment mais repris dans les plans, calcul
    ↪ de l'histogramme des coordonnées z
    count, value = np.histogram(murs[:,2], bins=50)
    max_z = value[np.amin(np.where(count == 0))]        #Le Z max du mur correspond à
    ↪ l'altitude à partir de laquelle il y a un "trou" dans les coordonnées

    bat = murs[murs[:,2]<max_z]                         #On ne garde dans le bâtiment
    ↪ que les points dont le z est inférieur au z max
    remains = np.concatenate((nuage, murs[~(murs[:,2]<max_z)]), axis=0)

    #Pour éliminer les points hors des murs (dans le prolongement des plans), création
    ↪ d'une grille 2D pour laquelle la hauteur min et max de chaque pixel est calculée
    #Sachant qu'un mur est vertical et fait au moins 2m, si la différence entre le min
    ↪ et le max est inférieure à 2, les points de ce pixel ne font pas partie du mur
    cell_size = 0.05
    grid = grid_2d(bat, cell_size)              #Création de la grille 2D

    i = 0
    for cell in grid:         #Parcours des pixels pour calculer la zmin et max

        z_min = np.amin(bat[grid[cell],2])     #Altitude minimale dans le pixel
        z_max = np.amax(bat[grid[cell],2])     #Altitude maximale dans le pixel

        if (z_max - z_min < 2):        #Si différence inférieure à 2, on retient les
        ↪ indices des points contenus dans le pixel
            if i ==0:
                ind_out = grid[cell]   #Si 1e cellule dont il faut rejeter les pts
                i +=1
            else :
                ind_out = np.concatenate((ind_out, grid[cell]))  #Sinon on ajoute les
                ↪ points à rejeter aux précédents

    out = np.zeros(len(bat), dtype=bool)
    out[ind_out] = True         #True si en-dehors des murs
    reste = bat[out]
    bat = bat[~out]             #On isole les points compris dans les murs
    bat[:,3] = 1                #Attribution de la classe 1 aux points des murs
    remains = np.concatenate((remains, reste), axis=0)      #On regroupe les points
    ↪ restants qui ne sont pas encore classés

    t4 = time.time()
    print('Bâtiments extraits en {:.3f} secondes'.format(t4 - t3))

#                               Portes
#-------------------------------------------------------------------------------

    nuage = remains             #On travaille sur le reste des points
    buffer = 2                  #Buffer de sécurité autour de la cible

    max_dist = 0.04             #Distance max au plan
    voisinage = brute_force_spherical(cible_door, nuage[:,0:3], radius, coord = True)
    ↪     #Voisinage des cibles des portes

    i = 0
    j = 0
    for key in voisinage:
        voisins = voisinage[i]              #Voisins de la cible traitée

        if len(voisins) >= 3:
            a,b,c,d = best_fitting_plane(voisins)           #Paramètres du plan
```

```
            #On ne garde les points que s'ils combinent deux conditions :
            #1) Situés à une distance < max_dist du plan ;
            #2) Situés à une distance < buffer de la cible
            if j == 0:
                in_plane = np.logical_and(plane(nuage[:,0:3], a,b,c,d, max_dist, unique
                ↪  = False), dist(cible_door[i], nuage[:,0:3],buffer))
                j += 1

            #Si pas 1e plan, on ajouter les nouveaux indices aux précédents
            else:
                in_plane = np.logical_or(in_plane, np.logical_and(plane(nuage[:,0:3],
                ↪  a,b,c,d, max_dist, unique = False), dist(cible_door[i],
                ↪  nuage[:,0:3],buffer)))
        i +=1

    portes = nuage[in_plane]      #Si True, point appartenant à une porte
    remains = nuage[~in_plane]    #Points restants devant encore être traités
    portes[:,3] = 2               #Attribution de la classe 2 aux points des portes
    t5 = time.time()

    print('Portes extraites en {:.3f} secondes'.format(t5 - t4))

#                             Fenêtres
#-------------------------------------------------------------------------------

    nuage = remains             #On travaille sur le reste des points
    buffer = 1                  #Buffer de sécurité
    max_dist = 0.04             #Distance max au plan
    voisinage = brute_force_spherical(cible_window, nuage[:,0:3], radius, coord = True)
    ↪       #Voisinage des cibles des fenêtres

    i = 0
    j = 0
    while i < len(voisinage):
        voisins = np.concatenate((voisinage[i], voisinage[i+1]), axis=0)       #On
        ↪  utilise les voisins des deux cibles traitées (correspondant à un plan)

        if len(voisins) >= 3:
            cibles_coord = cible_window[i:i+2,:]      #On garde les coordonnées des
            ↪  deux cibles traitées
            a,b,c,d = best_fitting_plane(voisins)     #Détermination des paramètres du
            ↪  plan s'ajustant au mieux
            in_plane = plane(nuage, a,b,c,d, max_dist, unique = False)    #Points dans
            ↪  le plan situés à une distance < max_dist

            xmin = np.amin(cibles_coord[:,0]) - buffer     #Coord minimale en X
            ↪  diminuée du buffer de sécurité
            xmax = np.amax(cibles_coord[:,0]) + buffer     #Coord max en X
            ymin = np.amin(cibles_coord[:,1]) - buffer     #Coord min en Y

            ymax = np.amax(cibles_coord[:,1]) + buffer     #Coord max en Y
            zmin = np.amin(cibles_coord[:,2]) - buffer     #Coord min en Z
            zmax = np.amax(cibles_coord[:,2]) + buffer     #Coord max en Z

            # Plusieurs conditions doivent être remplies par les points
            in_z = np.logical_and(nuage[:, 2] > zmin, nuage[:, 2] < zmax)
            ↪  #Renvoie 'True' pour les points du nuage compris dans l'intervalle z
            in_x = np.logical_and(nuage[:, 0] > xmin, nuage[:, 0] < xmax)
            ↪  #Renvoie 'True' pour les points du nuage compris dans l'intervalle x
            in_y = np.logical_and(nuage[:, 1] > ymin, nuage[:, 1] < ymax)
            ↪  #Renvoie 'True' pour les points du nuage compris dans l'intervalle y
            in_xy = np.logical_and(in_x, in_y)
            in_xyz = np.logical_and(in_xy, in_z)

            if j == 0:                                  #Si 1e objet
                box = np.logical_and(in_plane, in_xyz)  #Double condition
                j += 1
            else:
                box = np.logical_or(box, np.logical_and(in_plane, in_xyz))  #Ajout des
                ↪  points remplissant les conditions

        i += 2          #On traite les cibles 2 par 2 car 2 cibles pour un plan

    fenetres = nuage[box]    #Si True, point appartenant à une fenêtre
    remains = nuage[~box]    #Ou sinon, point restant
    fenetres[:,3] = 3        #Attribution de la classe 3 aux points des fenêtres

    t6 = time.time()
    print('Fenêtres extraites en {:.3f} secondes'.format(t6 - t5))
```

```
#                              Gouttières
#-------------------------------------------------------------------------------

    nuage = remains         #On travaille sur le reste des points
    radius = 0.03           #Rayon de recherche du voisinage pour les gouttières

    #Pour limiter la taille du nuage pour le calcul des normales, extraction des points
    ↪  situés à une certaine distance des cibles associées aux gouttières
    i = 0
    while i < len(cible_gout):
        if i == 0:
            box = dist(cible_gout[i,0:2], nuage[:,0:2],0.2)     #Pour chaque gouttière,
            ↪  extraction des points situés à une distance inférieure à 20cm
        else:
            box = np.logical_or(box,dist(cible_gout[i,0:2], nuage[:,0:2],0.2))  #On
            ↪  combine les sous-nuages autour de chaque gouttière
        i +=1

    sous_nuage = nuage[box]                 #Points proches des gouttières
    remains = nuage[~box]

    # On vérifie que le sous-nuage n'est pas vide (c'est-à-dire que les points font
    ↪  partie du nuage)
    if len(sous_nuage) > 0:
        normals = compute_normals(sous_nuage[:,0:3], radius)        #Calcul des
        ↪  normales des points du sous-nuage

        #Croissance de région pour extraire chaque gouttière du sous-nuage (renvoyant
        ↪  un booléen) en tenant compte des normales
        i = 0
        radius_reg = 0.02         #Rayon de recherche pour la croissance de région

        while i < len(cible_gout):
            if i == 0:
                region = RegionGrowing(cible_gout[i], sous_nuage[:,0:3],normals,
                ↪  radius_reg)   #RegionGrowing appliquée au départ de chaque cible
            else:
                region = np.logical_or(region, RegionGrowing(cible_gout[i],
                ↪  sous_nuage[:,0:3],normals, radius_reg))
            i +=1

        goutt = sous_nuage[region]      #Si True, point d'une gouttière
        goutt[:,3] = 4                  #Classe 4 pour les points des gouttières
        remains = np.concatenate((remains, sous_nuage[~region]), axis=0)

    # Si pas de gouttières dans le nuage
    else :
        goutt = np.array([], dtype=np.int64).reshape(0,4)

    t7 = time.time()
    print('Gouttières extraites en {:.3f} secondes'.format(t7 - t6))

    #On regroupe l'ensemble des points pour reformer le nuage complet classifié
    classif = np.concatenate((bat,goutt,portes,fenetres,remains), axis = 0)

    #  Ecriture des fichiers
    ########################

    t8 = time.time()

    #Ecriture des résultats au format .las
    laswrite(classif,'../Outputs/classif_cible_bat1.las', point_cloud.header)

    t9 = time.time()
    print('Fichier écrit en {:.3f} secondes'.format(t9 - t8))
    print('Temps total du script: {:.3f} secondes'.format(t9 - t0))
```

10.2. Algorithme de segmentation pour la méthode du levé

```
#-------------------------------------------------------------------------------
#
#                Script de segmentation/classification
#                  sur base d'un levé topographique
#
#-------------------------------------------------------------------------------

#  Importation des librairies et des fonctions
#**********************************************

import numpy as np
import time
import laspy as lp
from matplotlib import path

#-------------------------------------------------------------------------------
#
#                             Fonctions
#                          -------------

#Fonction renvoyant le segment créé sur base d'une Bbox (définie par les points levés)
#ou bien un tableau booléen disant si le point appartient (true) ou non à la Bbox
#Buffer pour étendre l'intervalle et ne pas rejeter des points compris dans l'élément
def Bbox(topo, nuage, buffer, unique = False, empty = False):

    #Détermination de l'indice du point bas du bat (zmin) pour l'exclure du polygone
    ind_min = np.where(topo[:,3] == np.amin(topo[:,3]))

    #Délimitation de la bounding box en z (sur base des coordonnées min et max)
    min_z = topo[ind_min[0],3] - buffer/4
    max_z = np.amax(topo[:,3]) + buffer/4
    in_z = np.logical_and(nuage[:, 2] > min_z, nuage[:, 2] < max_z)

    #Délimitation de la bounding box en x et y et vérification du sens du polygone (si
    ↪  sens anti-horloger, buffer positif pour agrandir, si sens horloger, négatif)
    poly = path.Path(np.delete(topo[:,1:3], ind_min, axis = 0))    #Polygone formé par
    ↪  les points levés délimitant l'intérieur ou l'extérieur du segment
    v = poly.vertices-poly.vertices[0,:]
    a = np.arctan2(v[1:,1],v[1:,0])
    if (a[1:] >= a[:-1]).astype(int).mean() <= 0.5:     #Si sens horloger, il faut
    ↪  changer le signe du buffer
        buffer = - buffer

    in_poly = poly.contains_points(nuage[:,0:2], radius = buffer)     #Renvoie 'true'
    ↪  pour les points du nuage contenu dans ce polygone ('false' sinon)

    #Création de la bounding box 3D comprenant les points satisfaisant les 2 conditions
    box = np.logical_and(in_poly, in_z)

    #Si on veut éliminer les points à une certaine distance à l'intérieur de la
    ↪  bounding box pour conserver uniquement les bords de l'élément
    if empty :
        bruit = poly.contains_points(nuage[:,0:2], radius = - 6*buffer)
        box = np.logical_and(box, np.logical_not(bruit))

    #Retourne l'objet segmenté et le reste du nuage de points
    if unique:
        segment = nuage[box]          #Points du nuage appartenant au segment
        remains = nuage[~box]         #Points restants
        return segment, remains

    #Retourne un tableau booléen distinguant les points du segment ou non
    else:
        return box

#-------------------------------------------------------------------------------

# Fonction utilisant la fonction Bbox pour extraire plusieurs objets du même type
def multi_Bbox(topo, nuage, buffer, empty = False):

    topo_div = np.split(topo, np.where(topo[:,5]==2)[0])     #Séparation des points
    ↪  quand le modificateur vaut 2 (signifiant le début d'un nouvel objet)
    topo_div = np.delete(topo_div,0,0)       #Suppression du 1e élément car rien avant
    ↪  le 1e objet (le 1e .2)

    i = 0
    while i < len(topo_div):
        if i == 0:
```

```
            if empty:
                box = Bbox(topo_div[i], nuage, buffer, empty = True)
            else:
                box = Bbox(topo_div[i], nuage, buffer, empty = False)
        else:
            if empty:
                box = np.logical_or(Bbox(topo_div[i], nuage, buffer, empty = True),box)
            else:
                box = np.logical_or(Bbox(topo_div[i], nuage, buffer, empty = False),box)
        i += 1

    segment = nuage[box]            #Points du nuage appartenant à la même classe
    remains = nuage[~box]           #Points restants
    return segment, remains

#-------------------------------------------------------------------------------

# Fonction permettant de déterminer les coordonnées des deux points délimitant un objet
↪ de type rectangulaire (extrémités diagonales) à partir de 3 pts mesurés
# A partir d'un tableau contenant les 3 points délimitant chaque objet, la fonction
↪ renvoie un tableau contenant les deux nouveaux points de chaque objet
def from3to2(topo):

    i = 0
    while i < len(topo):
        if topo[i][5] == 8:        #Le modificateur "8" signifie qu'il s'agit du point
        ↪ bien mesuré (au fond de l'objet)
            vect = topo[i][1:4] - topo[i-1][1:4]        #On calcule le vecteur qu'il
            ↪ faudra appliquer pour trouver le point manquant
            topo_2pts = topo[i]    #Ce pt doit être conservé pour extraire le segment

            #Selon que le point précédent ait été levé en 2e ou 3e position, le point à
            ↪ décaler varie
            if (i+2) % 3 == 0:
                pt2 = np.hstack((topo[i+1][0],topo[i+1][1:4]+vect,topo[i+1][4],[0]))
                ↪  #Le point créé doit avoir la même structure : ID, coordonnées
                ↪ (calculées en ajoutant le vecteur), code, modificateur
                topo_2pts = np.concatenate(([topo_2pts], [pt2]))      #On regroupe les
                ↪ deux points formant l'objet

            elif (i+1) % 3 == 0:
                pt2 = np.hstack((topo[i-2][0], topo[i-2][1:4]+vect, topo[i-2][4],0))
                ↪  #Le point créé doit avoir la même structure

                topo_2pts = np.concatenate(([topo_2pts], [pt2]))      #On regroupe les
                ↪ deux points formant l'objet
            else:
                print("Attention, il y a une erreur dans l'ordre des points mesurés!")

            if i<3:
                rect = topo_2pts     #2 points du 1e objet
            else:
                rect = np.concatenate((rect, topo_2pts))  #On ajoute ensuite les points
                ↪ des objets suivants
        i+=1

    return rect

#-------------------------------------------------------------------------------

# Fonction permettant de déterminer les coordonnées des deux points délimitant un objet
↪ de type rectangulaire (extrémités diagonales) à partir de 4 pts mesurés
# A partir d'un tableau contenant les 4 points délimitant chaque objet, la fonction
↪ renvoie un tableau contenant les deux nouveaux points de chaque objet

def from4to2(topo):

    topo_div = np.split(window_div[2], len(window_div[2])/4)   #On divise le tableau
    ↪ tous les 4 points pour séparer chaque objet

    #Boucle qui, pour chaque objet, va transformer les 4 points levés en 2 points
    i = 0
    while i <len(topo_div):
        pt1 = topo_div[i][3]    #Le dernier pt levé correspond à une des extrémités
        vect = topo_div[i][2][1:4]-topo_div[i][1][1:4]      #Le vecteur de déplacement
        ↪ correspond au vecteur entre les points levés 2 et 3
        pt2 = np.hstack((topo_div[i][0][0], topo_div[i][0][1:4]+vect,
        ↪ topo_div[i][0][4],[0]))       #Création du point définissant la 2e
        ↪ extrémité dont les coord sont celles du 1e pt levé déplacé du vecteur
        topo_2pts = np.concatenate(([pt1], [pt2]))         #On regroupe les 2 pts
        ↪ délimitant l'objet
```

```
        if i == 0 :
            rect = topo_2pts                #2 points du 1e objet
        else:
            rect = np.concatenate((rect, topo_2pts))    #On ajoute ensuite les points
            ↪ des objets suivants
        i += 1

    return rect

#-------------------------------------------------------------------------------

# Fonction utilisant deux points levés pour déterminer les points du nuage contenus
↪ dans le "rectangle" délimité par ces points et à une certaine distance de celui-ci
def rect(topo, nuage, buffer, dist, unique = False):
    p1 = topo[0,1:3]          #Coordonnées x et y du point levé n°1
    p2 = topo[1,1:3]          #Coordonnées x et y du point levé n°2
    p3 = nuage[:,0:2]         #Matrice reprenant les coord x et y des pts du nuage

    #Détermination des min et max du rectangle selon les trois coordonnées
    xmin = np.amin(topo[:,1]) - buffer
    xmax = np.amax(topo[:,1]) + buffer
    ymin = np.amin(topo[:,2]) - buffer
    ymax = np.amax(topo[:,2]) + buffer
    zmin = np.amin(topo[:,3]) - buffer
    zmax = np.amax(topo[:,3]) + buffer

    #Plusieurs cdts doivent être remplies par les pts pour faire partie du segment
    in_z = np.logical_and(nuage[:, 2] > zmin, nuage[:, 2] < zmax)     #Renvoie 'True'
    ↪ pour les points du nuage compris dans l'intervalle z
    in_x = np.logical_and(nuage[:, 0] > xmin, nuage[:, 0] < xmax)     #Renvoie 'True'
    ↪ pour les points du nuage compris dans l'intervalle x
    in_y = np.logical_and(nuage[:, 1] > ymin, nuage[:, 1] < ymax)     #Renvoie 'True'
    ↪ pour les points du nuage compris dans l'intervalle y
    in_xy = np.logical_and(in_x, in_y)
    in_xyz = np.logical_and(in_xy, in_z)       #Renvoie 'True' pour les points du nuage
    ↪ compris dans l'intervalle des trois coordonnées

    #Condition supp: on garde les pts situés à x cm du segment joignant les pts levés
    box = np.logical_and(np.cross(p2-p1, p1-p3)/np.linalg.norm(p2-p1) < dist, in_xyz)

    #Fonction retournant l'objet segmenté et le reste du nuage de points
    if unique:
        segment = nuage[box]        #Points du nuage appartenant au segment/objet
        remains = nuage[~box]       #Points restants
        return segment, remains

    #Fonction retournant un tableau booléen distinguant les points du segment ou non
    else:
        return box

#-------------------------------------------------------------------------------

# Fonction utilisant la fonction rect pour extraire plusieurs objets du même type
↪
def multi_rect(topo, nuage, buffer, dist):
    topo_div = np.split(topo, len(topo)/2)     #Séparation des points tous les 2 points
    ↪ (un rectangle étant délimité à chaque fois par 2 points successifs)

    # Boucle appliquant la fonction rect tant qu'il y a des points à traiter
    i = 0
    while i < len(topo_div):
        if i == 0:
            box = rect(topo_div[i], nuage, buffer, dist)   #Tableau booléen pour le 1e
            ↪ objet (True si point appartient à l'objet)
        else:
            box = np.logical_or(rect(topo_div[i], nuage, buffer, dist),box)
            ↪ #Combinaison du tableau booléen précédent avec celui qui est créé pour
            ↪ le nouvel objet
        i += 1
    return box

#-------------------------------------------------------------------------------

# Fonction extrayant un cylindre sur base de son centre, son rayon et sa hauteur
def cyl(topo, nuage, buffer, unique = False) :     #topo est un tableau contenant 3
↪ points: le centre bas, le point bas permettant de calculer le rayon et le point
↪ déterminant la hauteur

    rayon = np.linalg.norm(topo[0,1:3]-topo[1,1:3]) + buffer    #rayon du cylindre =
    ↪ distance entre le centre et le point pris sur le côté de la gouttière (+ajout
    ↪ d'un buffer de sécurité)
    centre = topo[0,1:3]              #coordonnées x et y du centre du cylindre
    h = topo[2,3]                     #hauteur du cylindre
```

```
    in_z = np.logical_and(nuage[:, 2] > topo[0,3], nuage[:, 2] < h)      #On garde les
    ↪  points compris entre le zmin et le zmax (h)
    box = np.logical_and(np.linalg.norm(centre-nuage[:,0:2], axis=1) < rayon, in_z)
    ↪  #On garde les points à une distance du centre inférieure au rayon (combiné avec
    ↪  la condition précédente)

    #Fonction retournant l'objet segmenté (cylindre) et le reste du nuage de points
    if unique:
        segment = nuage[box]        #Points du nuage appartenant au segment/objet
        remains = nuage[~box]       #Points restants
        return segment, remains
    #Fonction retournant un tableau booléen distinguant les points du segment ou non
    else:
        return box

#-------------------------------------------------------------------------------

# Fonction utilisant la fonction cyl pour extraire plusieurs objets du même type
def multi_cyl(topo, nuage, buffer):

    topo_div = np.split(topo, len(topo)/3)  #Séparation des pts tous les 3 pts,un
    ↪  cylindre étant délimité à chaque fois par 3 pts (centre, rayon, hauteur)

    # Boucle appliquant la fonction cyl tant qu'il y a des points à traiter
    i = 0
    while i < len(topo_div):
        if i == 0:
            box = cyl(topo_div[i], nuage, buffer)     #Tableau booléen pour le 1e objet
            ↪  (True si point appartient à l'objet)
        else:
            box = np.logical_or(cyl(topo_div[i], nuage, buffer),box)     #Combinaison
            ↪  du tableau booléen précédent avec celui créé pour le nouvel objet
        i += 1
    segment = nuage[box]        #Points du nuage appartenant au segment/objet
    remains = nuage[~box]       #Points restants
    return segment, remains

#-------------------------------------------------------------------------------

# Fonction écrivant les résultats (coordonnées x, y, z + classe) dans un fichier .las
def laswrite(nuage, filename, header):
    output = lp.file.File(filename, mode = "w", header = header)
    if(len(nuage)>0):
        output.x = nuage[:,0]
        output.y = nuage[:,1]
        output.z = nuage[:,2]
        output.raw_classification = nuage[:,3]
        print("Fichier .las enregistré !")
    else:
        print("Pas de points dans le nuage !")
    output.close()

#-------------------------------------------------------------------------------
#
#                                Main
#                               -------
if __name__ == '__main__':

    #  Importation et transformation des données
    ##########################################

   #Nuage de points au format .las
   t0 = time.time()
   point_cloud = lp.file.File('../Data/Scan/nuage_bat1_memoire.las')
   nuage = np.vstack((point_cloud.x, point_cloud.y, point_cloud.z)).T

   #Levé topographique (ID, X, Y, Z, code) au format .txt
   pts_topo = np.loadtxt('../Data/data_topo.txt',dtype = "str", delimiter=';')
   pts_topo = np.column_stack((pts_topo, np.zeros((len(pts_topo),1))))   #Ajout d'une
   ↪  colonne qui reprendra le modificateur

   t1 = time.time()
   print('Importation faite en {:.3f} secondes'.format(t1 - t0))

   #Boucle parcourant les points topos pour séparer les séquences/modificateurs
   i = 0
   while i < len(pts_topo):
       if '.' in pts_topo[i][4]: #Si liaison, séparation entre séquence et
       ↪  modificateur
           pts_topo[i][5] = pts_topo[i][4].split('.')[1]
           pts_topo[i][4] = pts_topo[i][4].split('.')[0]
       i += 1
   topo = pts_topo.astype(np.float)     #Conversion en float du tableau
```

```
#  Segmentation et classification du nuage de points
################################################

#Séparation des points selon leur code pour les regrouper par catégories

topo_bat = []              #Liste vide qui contiendra les points de bâtiments
topo_door = []             #Portes
topo_window = []           #Fenêtres
topo_gout = []             #Gouttières
topo_stat = []             #Stations
for i in topo:
    if 300 <= i[4] <= 305:
        topo_bat.append(i)
    elif 350 <= i[4] <=355:
        topo_door.append(i)
    elif 360 <= i[4] <= 365:
        topo_window.append(i)
    elif 408 <= i[4] <= 410:
        topo_gout.append(i)
    elif i[4] == 921:
        topo_stat.append(i)
    else:
        print("Code inconnu !")

topo_bat = np.asarray(topo_bat)
topo_door = np.asarray(topo_door)
topo_window = np.asarray(topo_window)
topo_gout = np.asarray(topo_gout)
topo_stat = np.asarray(topo_stat)

classif = np.zeros(len(nuage))          #Création d'une colonne de 0 qui définira la
↪  classe de chaque point
nuage = np.column_stack((nuage, classif))    #Ajout de cette colonne

t2 = time.time()
print('Prétraitements faits en {:.3f} secondes'.format(t2 - t1))

#                              Gouttières
#-------------------------------------------------------------------------------

if len(topo_gout) > 0 :
    buffer = 0.02     #Buffer de sécurité pour ne pas rejeter des points inclus
    goutt, remains = multi_cyl(topo_gout, nuage ,buffer)    #Extraction des
    ↪  gouttières du reste du nuage de points
    goutt[:,3] = 4       #Attribution de la classe 4 aux points des gouttières

    t3 = time.time()
    print('Gouttières extraites en {:.3f} secondes'.format(t3 - t2))

#                              Batiments
#-------------------------------------------------------------------------------

t4 = time.time()

buffer = 0.08          #Buffer de sécurité pour ne pas rejeter des points inclus
topo_bat_sorted = topo_bat[topo_bat[:, 4].argsort(kind = 'stable')]     #Tri sur
↪  base du code pour regrouper les points appartenant au même batiment
bat, remains = multi_Bbox(topo_bat_sorted, remains, buffer, empty = True)
↪  #Application de la fonction multi_Bbox pour extraire les points des bâtiments
bat[:,3] = 1          #Attribution de la classe 1 aux points des bâtiments

t5 = time.time()
print('Batiments extraits en {:.3f} secondes'.format(t5 - t4))

#                               Portes
#-------------------------------------------------------------------------------

t6 = time.time()

topo_door_sorted = topo_door[topo_door[:,4].argsort(kind = 'stable')]      #Tri des
↪  points pour les regrouper selon les mêmes codes
door_div = np.split(topo_door_sorted,
↪  np.where(topo_door_sorted[:-1,4]!=topo_door_sorted[1:,4])[0]+1)   #Séparation
↪  des portes levées en 2 ou en 3 pts (code différent)

#Traitement particulier à appliquer aux portes délimitées par 3 pts (une extrémité
↪  n'était pas visible)
#La fonction from3to2 transforme ces 3 pts en 2 pts correspondant aux extrémités
door_3pts = door_div[1]             #On traite les points mesurés en 3 pts
door = from3to2(door_3pts)
```

```
buffer = 0.02      #Buffer de sécurité sur les points mesurés
dist = 0.04        #Distance au plan pour tenir compte de l'épaisseur de la porte

doors = np.concatenate((door,door_div[0]))     #On regroupe toutes les portes
↪ ensemble pour appliquer la fonction multi-rect sur la totalité des portes levées
porte = multi_rect(doors, bat, buffer, dist)   #Extraction des portes du nuage
↪ contenant les batiments
bat[porte,3] = 2        #Attribution de la classe 2 aux points des portes

t7 = time.time()
print('Portes extraites en {:.3f} secondes'.format(t7 - t6))

#                              Fenêtres
#-------------------------------------------------------------------------

#Le même raisonnement doit être appliqué aux fenêtres puisque tous les points ne
↪ sont pas visibles sur le terrain
t8 = time.time()
topo_window_sorted = topo_window[topo_window[:,4].argsort(kind = 'stable')]
↪ #Tri des points pour les regrouper selon les mêmes codes
window_div = np.split(topo_window_sorted,
↪ np.where(topo_window_sorted[:-1,4]!=topo_window_sorted[1:,4])[0]+1)
↪ #Séparation des fenêtres levées en 2 ou en 3 pts (code différent)

#La fonction from3to2 transforme les 3 pts levés en 2 pts
window_3pts = window_div[1]           #On traite les points mesurés en 3 pts
window3 = from3to2(window_3pts)

#Certaines fenêtres ont été levées en 4 points avec un code différent suite à un
↪ appui de fenêtre penché
#Ces 4 points doivent être transformés en deux extrémités, grâce à la fct from4to2
window_4pts = window_div[2]
window4 = from4to2(window_4pts)

dist = 0.03             #Distance au plan

windows = np.concatenate((window_div[0], window3, window4))  #On regroupe toutes
↪ les fenêtres pour appliquer la fonction multi-rect sur la totalité des fenêtres
↪ levées
fenetres = multi_rect(windows, bat, buffer, dist)             #Extraction des
↪ fenêtres du nuage contenant les batiments (renvoie un booléen)
bat[fenetres,3] = 3      #Attribution de la classe 3 aux points des fenêtres

t9 = time.time()
print('Fenêtres extraites en {:.3f} secondes'.format(t9 - t8))

# On regroupe ensuite les points pour reformer le nuage entier classifié
classif = np.concatenate((goutt, bat, remains), axis = 0)

#  Ecriture des fichiers
#************************

#Ecriture des résultats au format .las
t10 = time.time()

laswrite(classif,'../Outputs/classif_topo_bat1.las', point_cloud.header )

t11 = time.time()
print('Fichier écrit en {:.3f} secondes'.format(t11 - t10))
print('Temps total du script: {:.3f} secondes'.format(t11 - t0))
```

10.3. Script de validation

```
#-------------------------------------------------------------------------------
#
#                Script de segmentation/classification
#                  sur base d'un levé topographique
#
#-------------------------------------------------------------------------------

#  Importation des librairies et des fonctions
#################################################

import numpy as np
import laspy as lp
from sklearn.metrics import confusion_matrix, accuracy_score

#-------------------------------------------------------------------------------
#
#                                  Main
#                                 --------
if __name__ == '__main__':

    #  Importation des données
    ############################

    # Nuages de points classifié au format .las
    cloud = lp.file.File('../Outputs/classif_cible_bat1.las')
    pc_classif = np.vstack((cloud.x, cloud.y, cloud.z, cloud.raw_classification)).T

    # Vérité terrain au format .las
    verite = lp.file.File('../Data/verite_terrain_bat1.las')
    pc_verite = np.vstack((verite.x, verite.y, verite.z, verite.raw_classification)).T
    print("Données chargées")

    #       Prétraitements
    ############################

    # Tri sur base des coordonnées pour comparer les classes des mêmes points
    pc_verite = pc_verite[np.lexsort(np.transpose(pc_verite)[::-1])]
    pc_classif = pc_classif[np.lexsort(np.transpose(pc_classif)[::-1])]
    print("Données triées")

    # On compare uniquement les classes/labels
    labels_classif = pc_classif[:,3:4]
    labels_verite = pc_verite[:,3:4]

    #         Validation
    ############################

    # Création de la matrice de confusion et détermination des VP, FP et FN
    cm = confusion_matrix(labels_verite, labels_classif)
    print(cm)

    VP = np.diag(cm)                    #Vrais positifs
    FP = np.sum(cm, axis=0) - VP        #Faux positifs
    FN = np.sum(cm, axis=1) - VP        #Faux négatifs

    # Overall Accuracy
    acc = accuracy_score(labels_verite, labels_classif)

    # Precision
    precision = VP/(VP + FP)

    # Recall
    recall = VP/(VP + FN)

    # F1-score
    F1_score = 2 * (precision * recall) / (precision + recall)

    print("accuracy:", acc)
    print("precision:", precision)
    print("recall:", recall)
    print("F1score:", F1_score)
```

10.4. Liste des cibles

11;734560.06;642191.742;248.292;99
15;734602.071;642193.193;248.089;99
1;734561.051;642154.005;247.378;99
2;734630.489;642216.672;248.046;99
100;734602.074;642193.195;248.286;99
101;734589.518;642224.272;248.21;3
102;734609.277;642205.158;248.042;3
103;734606.751;642192.543;247.633;99
104;734596.616;642186.116;246.922;99
105;734565.187;642175.448;248.133;3
106;734545.447;642194.539;248.161;3
107;734575.32;642182.318;248.075;3
108;734594.4;642201.98;248.014;3
109;734596.039;642203.381;247.047;1
110;734597.978;642204.287;246.95;99
111;734601.921;642204.278;246.78;99
112;734602.709;642211.406;247.135;2
113;734596.172;642211.24;248.295;99
114;734592.549;642206.714;247.326;2
115;734590.872;642208.319;247.343;2
116;734589.16;642210.141;247.055;1
117;734579.261;642204.847;248.167;99
118;734586.411;642215.156;248.607;99
119;734595.946;642217.969;247.276;2
120;734588.618;642224.825;247.117;1
121;734582.769;642213.395;248.305;99
122;734577.423;642219.634;248.056;3
123;734557.685;642199.168;248.091;3
124;734578.485;642220.369;247.044;1
125;734588.449;642209.191;248.134;3
126;734578.555;642199.481;246.795;99
127;734578.42;642203.711;246.935;99
128;734572.639;642214.136;246.796;99
129;734582.179;642202.859;247.315;2
130;734569.429;642204.145;248.188;99
131;734561.738;642187.757;247.722;99
132;734549.842;642190.392;247.272;2
133;734568.621;642188.75;248.122;3
134;734557.249;642183.411;247.967;4
135;734561.644;642179.167;247.796;4
136;734567.668;642188.061;246.97;1
137;734552.858;642187.645;248.005;4
138;734563.551;642188.142;248.325;99
139;734574.738;642177.835;246.934;99
140;734578.127;642172.059;248.404;99
141;734573.825;642174.549;248.11;99
142;734574.551;642181.31;247.092;1
143;734586.456;642188.431;246.838;99
144;734562.735;642171.481;247.483;2
145;734571.183;642184.735;247.15;2
146;734565.959;642174.777;246.993;1
147;734589.685;642183.896;248.192;99
148;734544.242;642195.844;246.990;1
149;734566.053;642174.913;247.693;4

10.5. Liste des points issus du levé topographique

5;734527.909;642187.596;246.636;921
4;734520.679;642237.221;246.481;921
3;734585.630;642254.136;247.174;921
2;734630.486;642216.672;246.752;921
7;734612.422;642104.210;245.289;921
1;734561.048;642154.004;245.882;921
10;734554.932;642214.162;246.650;921
11;734560.065;642191.742;246.796;921
12;734540.803;642242.655;246.762;921
13;734587.268;642218.633;246.669;921
14;734576.984;642236.066;246.687;921
15;734602.071;642193.194;246.795;921
100;734543.987;642196.234;249.578;300.2
101;734537.294;642189.327;249.552;300.4
103;734537.430;642189.334;246.706;300.4
102;734559.445;642167.931;249.584;300.4
104;734537.437;642189.270;246.657;408
105;734537.455;642189.226;246.657;409
106;734537.378;642189.265;249.502;410
107;734541.840;642185.009;246.669;408
108;734541.836;642184.978;246.668;409
109;734541.775;642185.017;249.517;410
110;734546.226;642180.768;246.688;408
111;734546.212;642180.731;246.688;409
112;734546.166;642180.767;249.526;410
113;734550.633;642176.523;246.693;408
114;734550.617;642176.490;246.693;409
115;734550.576;642176.528;249.538;410
116;734555.012;642172.295;246.683;408
117;734554.993;642172.263;246.683;409
118;734554.961;642172.282;249.531;410
119;734559.402;642168.065;246.683;408
120;734559.383;642168.033;246.683;409
121;734559.343;642168.046;249.534;410
122;734540.884;642192.892;248.612;361
123;734540.914;642192.822;248.618;361.8
124;734540.566;642192.543;249.204;361
125;734537.877;642188.939;247.680;361
126;734539.286;642187.544;248.856;361
127;734539.336;642187.613;248.864;361.8
128;734540.000;642186.835;248.878;361
129;734541.485;642185.436;247.698;361
130;734541.538;642185.505;247.702;361.8
131;734542.303;642184.692;248.881;361
132;734543.677;642183.313;247.695;361
133;734543.732;642183.378;247.701;361.8
134;734544.224;642182.797;246.828;350
135;734544.867;642182.159;248.869;350
136;734546.750;642180.340;249.204;361
137;734547.086;642179.999;248.623;361
138;734547.119;642180.078;248.625;361.8
139;734547.669;642179.460;249.214;361
140;734548.007;642179.126;248.628;361
141;734548.026;642179.205;248.630;361.8
142;734548.886;642178.311;248.903;361
143;734550.263;642176.952;247.719;361
144;734550.316;642177.021;247.723;361.8
145;734551.040;642176.210;248.905;361
146;734552.469;642174.832;247.711;361
147;734552.510;642174.903;247.715;361.8
149;734553.217;642174.101;248.891;361
150;734554.652;642172.711;247.709;361
151;734554.708;642172.777;247.714;361.8
152;734554.096;642201.290;246.846;350
153;734553.354;642201.960;248.898;350
154;734548.721;642223.461;248.919;360
155;734549.011;642223.125;248.357;360
156;734545.254;642226.516;249.764;301.2
158;734551.999;642206.884;247.747;361
159;734550.606;642205.431;248.935;361
160;734550.752;642205.297;248.941;361.8
161;734552.716;642207.626;248.929;361
163;734552.868;642207.487;248.939;361.8
162;734554.108;642209.071;247.752;361

164;734554.840;642209.840;248.934;361
166;734554.988;642209.699;248.944;361.8
165;734556.227;642211.275;247.752;361
167;734558.492;642213.315;247.754;360
168;734557.117;642211.891;248.945;360
169;734560.460;642215.664;248.927;361
171;734560.603;642215.526;248.942;361.8
170;734559.073;642214.219;247.753;361
172;734562.580;642217.872;248.929;361
174;734562.723;642217.728;248.936;361.8
173;734561.187;642216.420;247.749;361
175;734562.393;642218.555;247.751;361
177;734562.529;642218.702;247.753;361.8
176;734561.064;642220.044;248.938;361
178;734558.363;642222.396;247.749;361
179;734556.923;642223.783;248.920;361
180;734557.061;642223.905;248.939;361.8
181;734556.220;642223.952;247.757;361
182;734554.836;642222.661;248.940;361
183;734556.074;642224.089;247.755;361.8
184;734552.730;642220.304;247.755;361
185;734554.116;642221.747;248.936;361
186;734553.962;642221.882;248.946;361.8
187;734559.045;642221.961;248.938;350
188;734559.750;642221.281;246.849;350
189;734560.477;642220.596;246.847;350
190;734559.767;642221.281;248.933;350
191;734552.239;642219.795;249.759;301.4
192;734552.242;642219.812;246.706;301.4
193;734556.471;642224.218;249.767;301.4
194;734562.828;642218.145;249.770;301.4
195;734550.100;642204.951;249.768;301.4
196;734544.070;642196.126;246.670;408
197;734544.109;642196.106;246.671;409
198;734544.085;642196.154;249.525;410
200;734557.074;642198.226;249.760;301.4
201;734578.528;642220.408;249.772;301.4
202;734557.534;642198.767;247.756;361
203;734558.955;642200.141;248.919;361
204;734558.812;642200.274;248.937;361.8
205;734559.685;642200.892;247.761;361
206;734561.072;642202.334;248.923;361
207;734560.926;642202.463;248.939;361.8
208;734561.814;642203.117;247.762;361
209;734563.193;642204.534;248.925;361
210;734563.049;642204.670;248.941;361.8
211;734563.925;642205.277;248.941;361
212;734565.313;642206.730;247.755;361
213;734565.173;642206.868;247.755;361.8
214;734566.081;642207.561;248.943;361
215;734567.446;642208.936;247.763;361
216;734567.299;642209.066;247.754;361.8
217;734568.194;642209.733;248.935;361
218;734569.563;642211.130;247.754;361
219;734569.420;642211.264;247.752;361.8
220;734568.092;642188.491;247.668;362
221;734569.482;642189.926;247.660;362
222;734569.594;642189.811;247.724;362.8
223;734569.608;642189.796;248.899;362.8
224;734570.221;642190.684;247.662;362
225;734571.614;642192.123;247.670;362
226;734571.740;642192.000;247.739;362.8
227;734571.732;642191.993;248.912;362.8
228;734572.349;642192.881;247.664;362
229;734573.749;642194.320;247.665;362
230;734573.864;642194.197;247.729;362.8
231;734573.866;642194.198;248.883;362.8
232;734574.459;642195.072;247.662;362
233;734575.866;642196.515;247.666;362
234;734575.989;642196.390;247.734;362.8
235;734575.988;642196.392;248.905;362.8
236;734576.592;642197.273;248.270;362
237;734576.938;642197.636;248.274;362
238;734577.071;642197.501;248.329;362.8
239;734577.077;642197.502;248.895;362.8
240;734577.643;642198.353;248.256;362
241;734578.003;642198.714;248.268;362
242;734578.130;642198.588;248.330;362.8
243;734578.129;642198.592;248.891;362.8
244;734578.729;642199.470;248.262;362

245;734579.080;642199.829;248.273;362
246;734579.211;642199.703;248.329;362.8
247;734579.213;642199.708;248.897;362.8
248;734589.205;642224.514;248.897;360
249;734590.625;642223.122;247.739;360
250;734591.403;642222.399;248.893;360
251;734592.832;642221.021;247.741;360
252;734593.608;642220.275;248.895;360
253;734595.034;642218.912;247.737;360
254;734566.151;642174.830;249.578;300.4
255;734574.599;642181.248;249.740;300.2
256;734567.622;642187.996;249.745;300.4
258;734567.655;642187.981;246.728;300.4
257;734589.098;642210.162;249.743;300.4
259;734566.017;642174.880;246.674;408
260;734566.051;642174.892;246.674;409
261;734566.029;642174.925;249.527;410
262;734561.628;642179.102;246.658;408
263;734561.660;642179.107;246.658;409
264;734561.627;642179.172;249.522;410
265;734557.223;642183.405;246.833;408
266;734557.184;642183.419;246.833;409
267;734557.254;642183.408;249.527;410
268;734552.829;642187.632;246.833;408
269;734552.817;642187.665;246.831;409
271;734552.863;642187.645;249.539;410
272;734548.463;642191.871;246.837;408
273;734548.464;642191.919;246.837;409
274;734548.490;642191.881;249.539;410
276;734564.154;642176.604;247.710;361
277;734565.596;642175.232;248.872;361
278;734565.530;642175.176;248.879;361.8
279;734561.963;642178.751;248.892;361
280;734563.403;642177.344;247.707;361
281;734563.338;642177.286;247.712;361.8
282;734561.215;642179.472;248.881;361
284;734561.147;642179.406;248.886;361.8
283;734559.773;642180.855;247.716;361
285;734557.578;642182.969;248.879;361
286;734559.010;642181.590;247.715;361
287;734558.944;642181.527;247.718;361.8
288;734555.326;642185.028;247.731;360
289;734556.749;642183.644;248.893;360
290;734553.120;642187.148;248.893;360
291;734554.545;642185.770;247.730;360
292;734552.348;642187.908;248.888;361
293;734550.984;642189.345;247.714;361
294;734550.927;642189.275;247.720;361.8
295;734548.810;642191.414;248.874;350
296;734550.183;642190.090;246.846;350
297;734548.066;642192.191;248.898;361
298;734546.597;642193.587;247.721;361
299;734546.540;642193.520;247.726;361.8
300;734545.755;642194.320;248.879;361
301;734544.411;642195.696;247.707;361
302;734544.355;642195.631;247.713;361.8
303;734571.517;642184.392;246.826;350
304;734570.850;642185.010;248.879;350
400;734580.140;642200.945;248.280;362
401;734579.783;642200.577;248.278;362
402;734579.921;642200.451;248.338;362.8
403;734579.932;642200.462;248.925;362.8
404;734582.410;642203.272;248.927;351
405;734581.722;642202.554;246.798;351
406;734581.805;642202.464;246.803;351.8
407;734584.380;642205.321;247.673;362
408;734582.985;642203.876;247.670;362
409;734583.120;642203.751;247.736;362.8
410;734583.124;642203.748;248.910;362.8
411;734586.505;642207.516;247.673;362
412;734585.114;642206.078;247.673;362
413;734585.248;642205.954;247.737;362.8
414;734585.251;642205.949;248.907;362.8
415;734588.631;642209.709;247.677;362
416;734587.239;642208.266;247.672;362
417;734587.374;642208.144;247.737;362.8
418;734587.379;642208.134;248.917;362.8
419;734591.340;642208.043;246.816;351
420;734591.240;642207.974;246.814;351.8
421;734589.900;642209.409;248.909;351

422;734592.258;642207.128;248.918;351
423;734592.962;642206.488;246.812;351
424;734592.868;642206.411;246.816;351.8
425;734571.683;642213.336;248.936;361
426;734570.294;642211.895;247.762;361
427;734570.162;642212.034;247.756;361.8
428;734573.812;642215.531;247.761;361
429;734572.413;642214.088;248.916;361
430;734572.277;642214.233;248.941;361.8
431;734575.919;642217.728;247.761;361
432;734574.543;642216.288;248.930;361
433;734574.402;642216.429;248.946;361.8
434;734578.054;642219.921;247.764;361
435;734576.657;642218.491;248.927;361
436;734576.521;642218.631;248.946;361.8
437;734595.501;642218.201;246.803;351
438;734596.225;642217.513;248.910;351
439;734596.306;642217.607;248.930;351.8
440;734597.859;642215.928;248.272;362
441;734598.207;642215.581;248.270;362
442;734598.332;642215.717;248.341;362.8
443;734598.332;642215.717;248.921;362.8
444;734598.936;642214.878;248.272;362
445;734599.290;642214.534;248.271;362
446;734599.414;642214.666;248.340;362.8
447;734599.418;642214.666;248.925;362.8
448;734600.039;642213.798;248.273;362
449;734600.401;642213.444;248.270;362
450;734600.519;642213.588;248.341;362.8
451;734600.520;642213.586;248.938;362.8
452;734601.118;642212.756;248.273;362
453;734601.481;642212.405;248.272;362
454;734601.605;642212.540;248.341;362.8
455;734601.611;642212.530;248.935;362.8
456;734568.745;642237.290;248.937;360
457;734567.368;642235.861;247.762;360
458;734569.515;642238.040;247.761;360
459;734570.868;642239.488;248.925;360
500;734588.556;642224.882;249.742;302.2
501;734595.265;642231.896;249.751;302.4
502;734571.570;642227.164;249.773;301.4
503;734567.312;642222.758;249.769;301.4
504;734590.102;642226.325;248.903;350
505;734589.408;642225.628;246.807;350
506;734572.370;642226.396;246.843;351
507;734573.095;642225.707;248.923;351
508;734572.984;642225.617;248.933;351.8
509;734571.069;642226.631;248.945;361
510;734569.702;642225.214;247.755;361
511;734569.843;642225.082;247.754;361.8
512;734568.973;642224.445;248.935;361
513;734567.579;642223.019;247.754;361
514;734567.718;642222.887;247.754;361.8
515;734566.740;642223.113;248.927;350
516;734565.378;642224.556;246.852;350
517;734563.290;642226.407;246.842;350
518;734564.723;642225.043;248.921;350
519;734561.372;642228.431;248.916;351
520;734561.285;642228.331;248.925;351.8
521;734562.810;642227.043;246.842;351
522;734562.547;642230.562;248.932;361
523;734561.172;642229.129;247.757;361
524;734561.030;642229.268;247.758;361.8
525;734564.653;642232.771;247.758;361
526;734563.277;642231.324;248.914;361
527;734563.135;642231.461;248.936;361.8
528;734566.788;642234.973;247.751;361
529;734565.391;642233.531;248.924;361
530;734565.249;642233.669;248.946;361.8
531;734571.610;642240.256;248.941;360
532;734572.989;642241.690;247.754;360
550;734560.926;642228.876;249.766;301.4
551;734573.608;642242.065;249.760;301.4
552;734566.629;642248.777;249.762;301.4
553;734569.772;642245.781;248.917;351
554;734569.669;642245.666;248.921;351.8
555;734570.493;642245.081;246.703;351
556;734597.965;642229.324;247.677;362
557;734599.421;642227.916;247.683;362
558;734599.288;642227.792;247.747;362.8

559;734599.290;642227.789;248.909;362.8
560;734600.184;642227.196;247.674;362
561;734601.633;642225.798;247.678;362
562;734601.503;642225.673;247.741;362.8
563;734601.506;642225.667;248.900;362.8
564;734602.370;642225.103;247.672;362
565;734603.833;642223.689;247.672;362
566;734603.699;642223.565;247.736;362.8
567;734603.712;642223.557;248.892;362.8
570;734547.118;642228.467;247.754;361
571;734545.739;642227.026;248.911;361
572;734545.884;642226.892;248.924;361.8
573;734549.226;642230.683;247.750;361
574;734547.849;642229.225;248.918;361
575;734547.994;642229.096;248.927;361.8
576;734551.471;642232.738;247.755;360
577;734550.107;642231.304;248.930;360
578;734552.200;642233.506;247.753;360
579;734553.569;642234.926;248.927;360
580;734554.175;642235.839;247.752;361
581;734555.567;642237.282;248.919;361
582;734555.705;642237.139;248.927;361.8
583;734556.296;642238.032;247.761;361
584;734557.680;642239.474;248.930;361
585;734557.818;642239.331;248.939;361.8
586;734558.423;642240.239;247.762;361
587;734559.795;642241.685;248.912;361
588;734559.929;642241.542;248.922;361.8
589;734560.519;642242.439;247.756;361
590;734561.914;642243.880;248.913;361
591;734562.056;642243.734;248.924;361.8
592;734564.742;642246.826;247.752;361
593;734566.147;642248.264;248.917;361
594;734566.285;642248.122;248.928;361.8
599;734617.499;642210.413;246.778;302.4
600;734617.501;642210.425;249.744;302.4
601;734610.793;642203.458;249.749;302.4
602;734614.495;642207.296;248.931;351
603;734613.792;642206.576;246.821;351
604;734613.705;642206.651;246.822;351.8
605;734617.016;642210.903;247.671;362
606;734615.578;642212.302;247.681;362
607;734615.457;642212.167;247.747;362.8
608;734615.464;642212.171;248.907;362.8
609;734614.829;642213.038;247.679;362
610;734613.387;642214.432;247.673;362
611;734613.257;642214.292;247.744;362.8
612;734613.255;642214.295;248.922;362.8
613;734612.627;642215.160;247.670;362
614;734611.184;642216.559;247.676;362
615;734611.062;642216.415;247.741;362.8
616;734611.060;642216.419;248.908;362.8
617;734610.428;642217.282;247.674;362
618;734608.992;642218.683;247.673;362
619;734608.870;642218.543;247.743;362.8
620;734608.868;642218.549;248.899;362.8
621;734608.242;642219.416;247.665;362
622;734606.800;642220.802;247.672;362
623;734606.670;642220.672;247.742;362.8
624;734606.662;642220.672;248.887;362.8
625;734606.044;642221.542;247.669;362
626;734604.594;642222.940;247.668;362
627;734604.480;642222.797;247.737;362.8
628;734604.475;642222.805;248.898;362.8
629;734597.216;642230.035;247.678;362
630;734606.805;642226.733;247.845;362
631;734595.656;642231.283;247.743;362.8
632;734595.629;642231.295;248.901;362.8
650;734596.096;642203.448;249.745;300.4
651;734602.253;642211.675;248.903;351
653;734602.345;642211.765;248.911;351.8
652;734603.679;642210.270;246.808;351
654;734605.874;642208.154;247.676;362
655;734604.438;642209.548;247.679;362
656;734604.576;642209.671;247.747;362.8
657;734604.576;642209.668;248.915;362.8
658;734608.068;642206.035;247.671;362
659;734606.631;642207.427;247.681;362
660;734606.764;642207.551;247.749;362.8
661;734606.767;642207.549;248.913;362.8

662;734610.265;642203.908;247.683;362
663;734608.827;642205.300;247.675;362
664;734608.966;642205.422;247.744;362.8
665;734608.967;642205.424;248.919;362.8
666;734594.227;642201.512;247.676;362
667;734595.629;642202.961;247.673;362
668;734595.495;642203.080;247.739;362.8
669;734595.484;642203.077;248.910;362.8
670;734593.506;642200.766;247.665;362
671;734592.110;642199.320;247.668;362
672;734591.987;642199.458;247.737;362.8
673;734591.990;642199.443;248.908;362.8
674;734591.381;642198.566;247.668;362
675;734589.988;642197.128;247.672;362
676;734589.858;642197.261;247.739;362.8
677;734589.860;642197.255;248.913;362.8
678;734589.255;642196.358;247.671;362
679;734587.853;642194.921;247.672;362
680;734587.731;642195.062;247.745;362.8
681;734587.725;642195.059;248.907;362.8
682;734587.126;642194.169;247.666;362
683;734585.727;642192.731;247.700;362
684;734585.607;642192.861;247.743;362.8
685;734585.608;642192.861;248.907;362.8
686;734584.996;642191.980;247.667;362
687;734583.593;642190.544;247.667;362
688;734583.473;642190.678;247.737;362.8
689;734583.470;642190.675;248.907;362.8
690;734582.863;642189.773;247.670;362
691;734581.466;642188.334;247.672;362
692;734581.339;642188.465;247.742;362.8
693;734581.334;642188.461;248.911;362.8
694;734580.746;642187.579;247.674;362
695;734579.339;642186.128;247.675;362
696;734579.218;642186.268;247.740;362.8
697;734579.240;642186.296;248.910;362.8
698;734578.622;642185.386;247.675;362
699;734577.218;642183.947;247.669;362
700;734577.091;642184.078;247.736;362.8
701;734577.132;642184.132;248.889;362.8
702;734576.482;642183.198;247.667;362
703;734575.078;642181.749;247.669;362
704;734574.960;642181.890;247.737;362.8
705;734574.954;642181.875;248.905;362.8
720;734563.062;642171.818;248.910;350
721;734562.411;642171.143;246.832;350
722;734559.044;642168.492;247.718;361
723;734557.602;642169.861;248.882;361
724;734557.669;642169.916;248.899;361.8
725;734556.849;642170.618;247.715;361
726;734555.416;642171.972;248.879;361
727;734555.484;642172.026;248.896;361.8
728;734546.020;642181.032;248.620;361
729;734545.650;642181.389;249.188;361
730;734545.717;642181.443;249.205;361.8