Deliverable Title

Document type	**Deliverable T3.2-T3.3.** Computational algorithms developed in T3.2 and T3.3.
Document Version / Status	**2.0 - First issue**
Primary Authors	Rosa María Túñez Alcalde, Lucía Díaz Vilariño
Distribution Level	**PU (Public)**
Project Acronym	**RecycleBIM**
Project Title	**Integrated Planning and Recording Circularity of Construction Materials through Digital Modelling**
Grant Agreement Number	**101003575**
Project Website	**https://recyclebim.eu/**
Project Coordinator	**Miguel Azenha**
Document Contributors:	**Rosa María Túñez Alcalde, Muataz Safaa Abed Albadri, Antonio Fernández, Lucía Díaz Vilariño**

Change log

History of changes				
Version	**Date**	**Author (Organization)**	**Change**	**Page**
1.00	31/10/2023	R.M. Túñez (UVigo)	Creation of the document	all
1.10	15/11/2023	L. Díaz-Vilariño (UVigo)	Internal review	all
1.2	30/05/2024	R.M. Túñez (UVigo)	Document update	all

Table of Contents

List of figures

1. Introduction

The purpose of this document is to briefly explain the general structure of the code developed within T3.2 and T3.3. Functions included in the code are devoted to the *preprocessing* and *semantic segmentation* of building point clouds. *Preprocessing* module includes functionalities of *subsampling*, *denoising* and *removing outdoor points*, while *semantic segmentation* module is focused on point cloud segmentation at three different spatial scales: *floorplan or storey*, *room/space*, and *building element type*. Up to now, the automatically detected classes are *floor*, *ceiling*, *wall*, *doors,* and *the intersection points of the walls within each room along with their adjacency* are parametrized. However, the code is still under continuous improvement, and other element types such as windows and columns will be further considered. Figure 1 shows a schema of the code general structure of T3.2. developments.

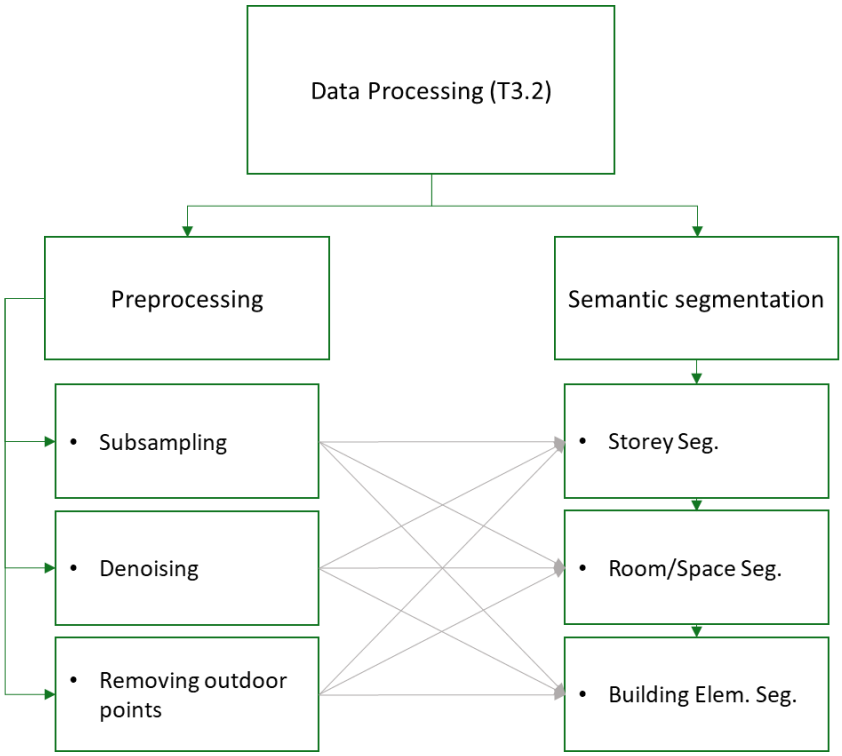


Figure 1. General Structure of the code developed in T3.2.

2. Structure of the General Folder

According to the RecycleBIM proposal, computational algorithms developed in T3.2. and T3.3. are uploaded to GitHub for global sharing. The folder containing the code is structured into five subfolders: *test* (main program), *data* (input data), *parameters* (the parameters that are needed), *src* (source code or functions) and *results* (outputs from processing input data). However, due to space restrictions just *test* and *src* were uploaded to GitHub, together with *License* and *Readme*. *Data* and *results* are uploaded to the UVigo data repository and shared through the link: https://dpv.uvigo.es/index.php/s/d64FQQE4sk3Z3yT

3. Test Folder

This folder contains the main program **main_cs1.py** that corresponds to a storey of the School of Mining and Energy Engineering at the University of Vigo. This main program includes the optimal parameters to process this case study, and their corresponding provided in *results* folder (see Section 5. Data Folder).

4. Parameters Folder

This folder contains the **parameters_cs1.py** file which contains the parameters required for code execution.

5. SRC Folder

This folder is organized into five subfolders: (1) adjacency, (2) *elements_seg*, (2) *morph_seg*, (4) *preprocessing*, (5) *storey_seg* and (6) *utils*.

1. *adjacency* in composed by two functions that correspond to the adjacency of the walls.
    - **intersection_points_index.py:** calculates points of intersection of walls.
    - **points_room_order.py:** arrange the intersection points within each room.
2. *elements_seg* is composed of three functions which correspond to the detection of building element type such as doors, walls, ceiling, and floor.
    - **doors_v2.py**: door detection process.
    - **extract_floor_ceiling.py**: ceiling and floor detection process.
    - **walls_index.py**: walls detection process.
3. *morph_seg* contains five functions that constitute the morphological segmentation of the building's floors.
    - **dilation.py**: morphological dilation adds voxels according to the shape and size of the structuring element.
    - **erosion.py**: morphological erosion is applied to empty voxels to break the space continuity between rooms given by doors.
    - **individualization.py**: remaining voxels after erosion operation are clustered on basis connectivity between voxels.
    - **occupied_voxels_classification.py**: occupied voxels are labelled regarding proximity of labelled empty voxel.
    - **point_cloud_classification.py**: reclassifies voxels in the point cloud.
4. *preprocessing* is made up of two functions for cleaning and voxelization (if necessary) of the point cloud.
    - **clean_exteriors.py**: clean point cloud from exterior parts. Uses DBSCAN to achieve the task.
    - **subsampling.py**: apply voxel grid filter and keep original order of points.
5. *storey_seg* is made up of functions that will divide the point cloud in storeys, being able to use the trajectory or not.
    - **histogram.py**: divide the point cloud according to the histogram.
    - **Histogram_scott.py**: divide the point cloud according to the histogram using Scott's rule.
    - **pointcloudpy.py**: reads the point cloud and trajectory.
    - **storeydivision.py**: divide the point cloud based on the trajectory.
    - **trajectorypy.py**: contains trajectory information and provides methods for working with it.
6. *utils* is made up of various functions that are necessary for the correct execution of the code, such as loading the point cloud, changing from numpy array to open3d, normal calculation, among others.
    - **array_to_o3d.py**: convert a numpy array to a point cloud.
    - **bounding_polygon_filter.py**: apply bounding polygon filter.
    - **compute_normals.py**: calculate normals[1] on a set of three-dimensional points.

[1] The term 'normals' refers to the normal of the plane containing a point neighbourhood.

- **functions_geom.py**: functions for geometric calculations.
- **functions_walls.py**: functions to detect walls.
- **functionspy.py**: functions to carry out storey segmentation.
- **get_grid_index.py**: calculated index.
- **grid_dataframe_to_3Dimage.py**: generate 3D image from voxelized data.
- **image3D_to_grid_dataframe.py**: generate voxelized data from image 3d.
- **load_file.py**: load point clouds in general.
- **load_pointcloud.py**: is ready to load the point clouds of the cases studied.
- **mix_room.py**: applies if the rooms were poorly divided.
- **remove_no_continuous.py**: remove non continuous celling.
- **search_indexes.py**: search the indexes in a dataframe.

6. Data Folder

This folder contains the input data of one case study which corresponds to a storey of the School of Mining and Energy Engineering at the University of Vigo (Figure 2).

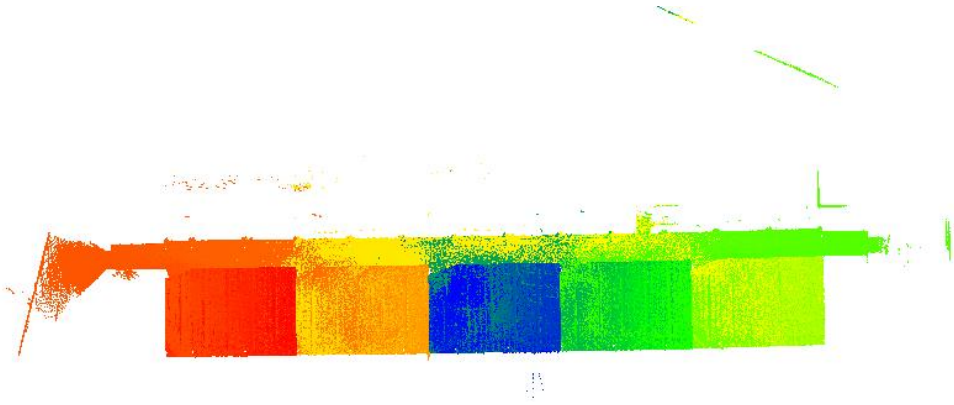


Figure 2. Input data of Case study 1.

7. Results Folder

Results contain the outputs of processing the input data using the parameters included in the **parameters_cs1.py** (*parameters* folder). In subfolder cs1, the following files are included:

- If the point cloud only has one storey, the result will be a file in txt format with the previous attributes (i.e. x, y, z, r, g, b, i, etc.) and new attributes such as 'id_storey', 'id_room', 'id_element', 'id_entity'.
- Execution time.
- A folder with results containing information about walls.

8. Requirements and Dependencies

The libraries necessary for the correct execution of the code are included hereafter:

Time (https://docs.python.org/es/3/library/time.html)

Logging (https://docs.python.org/es/3/library/logging.html)

Numpy (https://numpy.org/)

Open3D (http://www.open3d.org/)

Pandas (https://pandas.pydata.org/)

Laspy (https://laspy.readthedocs.io/en/latest/)

Alphashape (https://alphashape.readthedocs.io/en/latest/)

Itertools (https://docs.python.org/es/dev/library/itertools.html)

Matplotlib (https://matplotlib.org/)

Os(https://docs.python.org/es/3.10/library/os.html)

Cc3d (https://pypi.org/project/connected-components-3d/)

Scipy.spatial.cKDTree
(https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html)

Scipy.stats.mode (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mode.html)

Scipy.ndimage.binary_erosion
(https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.binary_erosion.html)

Scipy.ndimage.binary_dilation(https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.binary_dilation.html)

Scipy.spatial.distance.cdist
(https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html)

Scipy.signal.find_peaks
(https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html)

9. Instructions for use

As mentioned in Section 7, the correct execution of the code requires from installing several libraries. Additionally, paths need to be indicated by the user. A brief clarification if provided hereafter.

- `path (str)`: path of the data folder.
- `folder_cloud (str)`: path of the case study.
- `folder_results (str)`: path where the results will be saved.
- `name_cloud (str)`: name of the point cloud file.
- `name_trajectory (str)`: name of the trajectory file.
- `name_f (str)`: name to save the output file.

The code is prepared to save intermediate results and output files are related with the process executed for obtaining those results.

9.1. Case Study 1

9.1.1. Parameters

```
voxel_size = 0.1
eps = 0.3
pt = 2
nt = 0.9
alpha = 3.2
ransac_th = 0.12
zmax = 1000
zmin = -1000
dist = 0.1
pcd_in = None
se = None
vox_idx = np.array([])
clean_idx = np.array([])
vox_labels = pd.DataFrame()
empty_in_lbl = 1
occ_lbl = 10
width_door = 1.5
pts_door = 2.5
eps_door = 2
min_pt_door = 2
min_ratio = 0.05
threshold = 0.09
iterat = 2000
init_n = 3
h = 2.5
max_distance = 0.5
min_points_to_save = 3
dist_pl = 2.5
min_angle = 0
max_angle = 40
dist_centroids_1 = 5
b_f = 0.5
```

```
min_adjacent_pairs = 6
max_buffer_radius = 3
dist_centroids_2 = 4
ex_v = [1]
min_p_int = 4
max_dist_int = 8
min_angle_d = 85
max_angle_d = 95
perfect_angle = 90
flat_angle = 180
angle_points_global = 20
dist_max = 0.05
images = False
write = False
```

9.1.2. Implementation

To initiate the processing of case study 1, the following paths need to be established.

```
path = "C:\\...\\Code\\data"
folder_cloud = "\\cs1"
folder_results = "C:\\...\\Code\\results"
name_cloud = "\\cs1_storey_0.txt"
name_f = "_pointcloud_0_01.txt"
```

In this case, the point cloud is in txt format whose attributes are `x`, `y`, `z` and `time` (Figure 3). Furthermore, the point cloud is composed of one storey. Accordingly, `'id_storey'` is set to '0', by default.

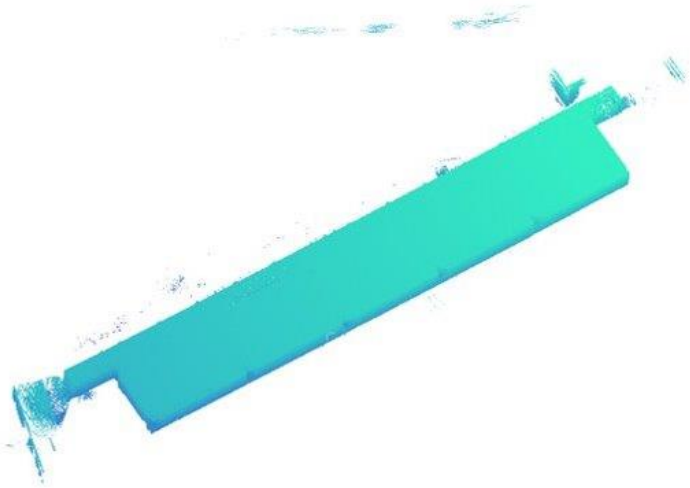


Figure 3. Initial point cloud.

Once the different segmentation functions are applied, a data frame is obtained with the format shown in Figure 4. The attributes that make up the data frame in Figure 10 follow the scheme shown in Section 10.

```
In [27]: class_pcd_df
Out[27]:
                x          y       z          time id_storey id_room id_element
0         -2.5144   0.377200  0.1177  1.559834e+09        00      01         02
1         -2.5287   0.379500  0.1471  1.559834e+09        00      01         02
2         -2.5101   0.376700  0.1746  1.559834e+09        00      01         02
3         -2.5100   0.376800  0.2031  1.559834e+09        00      01         02
4         -2.5134   0.377400  0.2321  1.559834e+09        00      01         02
...           ...        ...     ...           ...       ...     ...        ...
32451998 -9.2528 -20.389400 -1.7150  1.559835e+09        00      02         04
32451999 -9.2408 -20.419701 -1.7207  1.559835e+09        00      02         04
32452000 -9.2121 -20.436701 -1.7130  1.559835e+09        00      02         04
32452001 -9.2016 -20.468500 -1.7199  1.559835e+09        00      02         04
32452002 -9.1835 -20.494301 -1.7208  1.559835e+09        00      02         04

[32452003 rows x 7 columns]
```

Figure 4. Point cloud Data Frame format.

If the data is displayed depending on the attribute, the results shown in Figure 5 can be seen.

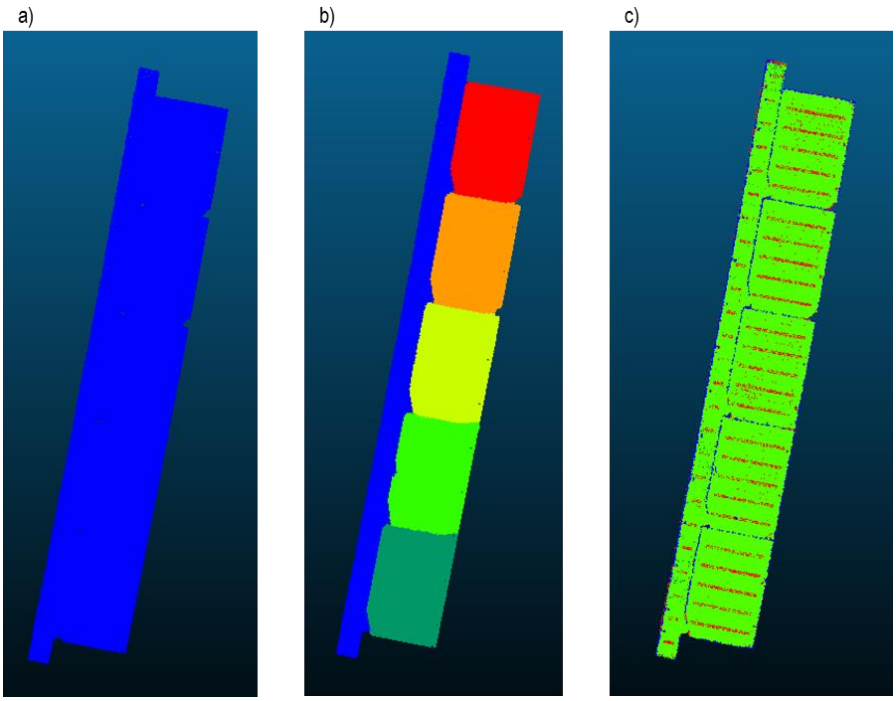


Figure 5. Data visualizated based on a) id_storey attribute, b) id_room attribute, c) id_element attribute.

The walls of the different rooms shown in Figure 6.

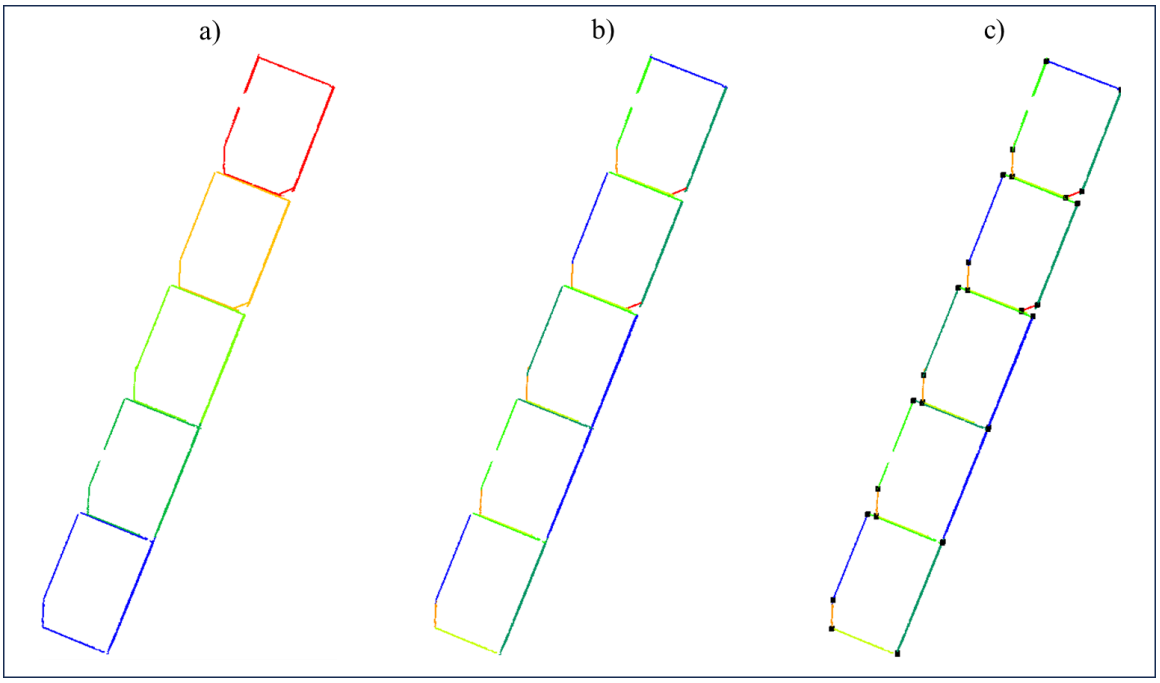


Figure 6. a) Walls by room, b) each wall in each room, c) points of intersection.

Figure 7 show intersection in the floor and in the ceiling.

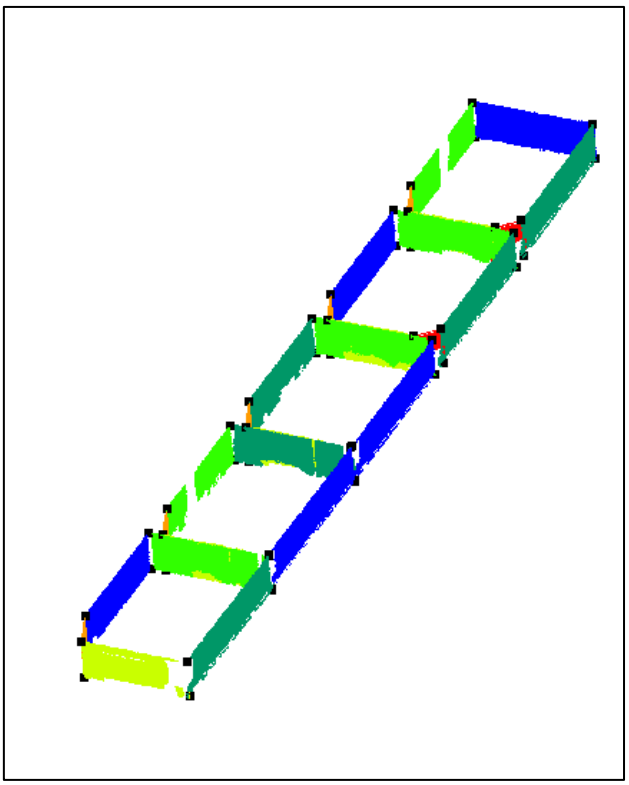


Figure 7. Points intersection in the floor and in the ceiling.

10. Additional notes

The code has been developed and tested in Python 3.9. Please note that the code may not function as expected with other versions of Python and could result in errors or unexpected behaviour. It is strongly recommended to use Python 3.9 or a compatible version to ensure proper code functioning.

In order do display in CloudCompare the results, the following considerations need to be taken:

(1) Red, Green and Blue default attributes (Figure 8) need to be changed by Scalar (Figure 9).

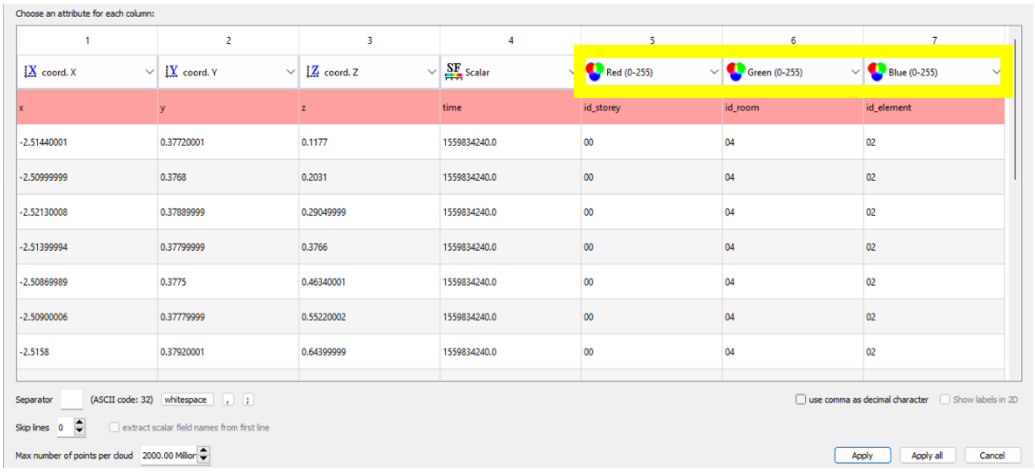


Figure 8. Red, Green, and Blue default attributes.

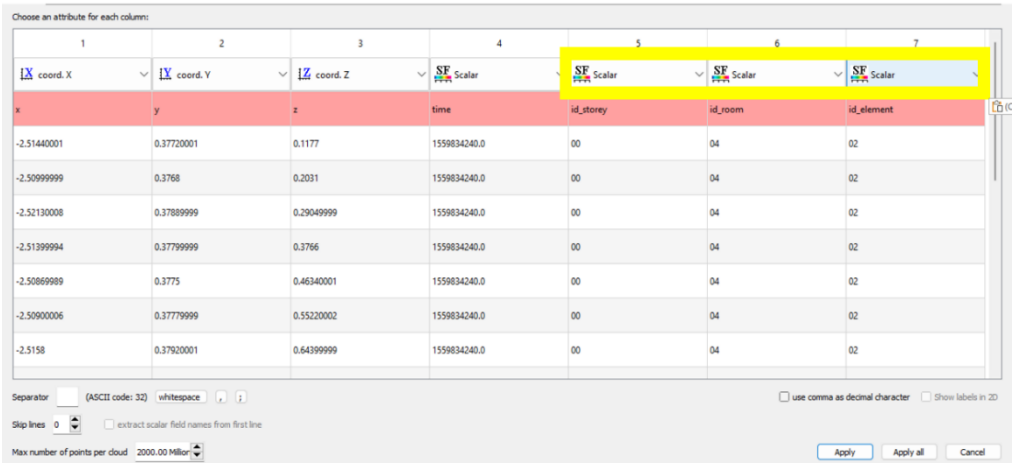
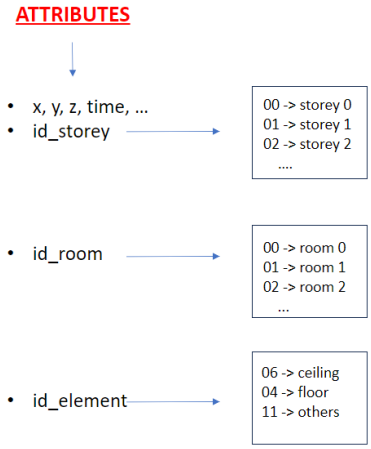


Figure 9. Scalar attributes.

- To execute the room/space segmentation process, the data frame of point cloud must have the *id_storey* attribute.
- There are two functions created to read point clouds. One of them (**load_pointcloud.py**) is ready for these case studies, and in the other (**load_file.py**) you must indicate if the file has a header and if it does not, write the attributes.
- Attributes:



Class	Label
beam	01
wall	02
column	03
slab/flooring	04
foundation	05
ceiling	06
stairs	07
railings	08
doors	09
Windows/curtain walls	10
others	11

Figure 10. Composition of attributes.

11. Conclusions

This document is intended to provide formal documentation of the code developed in T3.2 and T3.3. The code is fully tested in one case study. All the code is available in the GitHub repository at the following link: https://github.com/GeoTechUVigo/RecycleBIM_WP3 and *Data* and *results* are uploaded to the UVigo data repository and shared through the link: https://dpv.uvigo.es/index.php/s/d64FQQE4sk3Z3yT.

Significant improvements will be made to the code over up to M30, with the aim of optimizing its efficiency, time, and robustness. This improvement process will be carried out through continuous reviews, tests, and the implementation of other types of element's detections such as windows, columns, among others.