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, Patricia González Cabaleiro, Antonio Fernández, Lucía Díaz Vilariño**

Change log

History of changes				
Version	**Date**	**Author (Organization)**	**Change**	**Page**
1.0	31/10/2023	R.M. Túñez (UVigo)	Creation of the document	all
1.1	15/11/2023	L. Díaz-Vilariño (UVigo)	Internal review	all
1.2	30/05/2024	R.M. Túñez (UVigo)	Document update	all
1.3	31/04/2025	R.M.Túñez (UVigo)	Document update	all
1.4	05/05/2025	L. Díaz-Vilariño (UVigo)	Internal review	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, T3.3 and T3.4. 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.

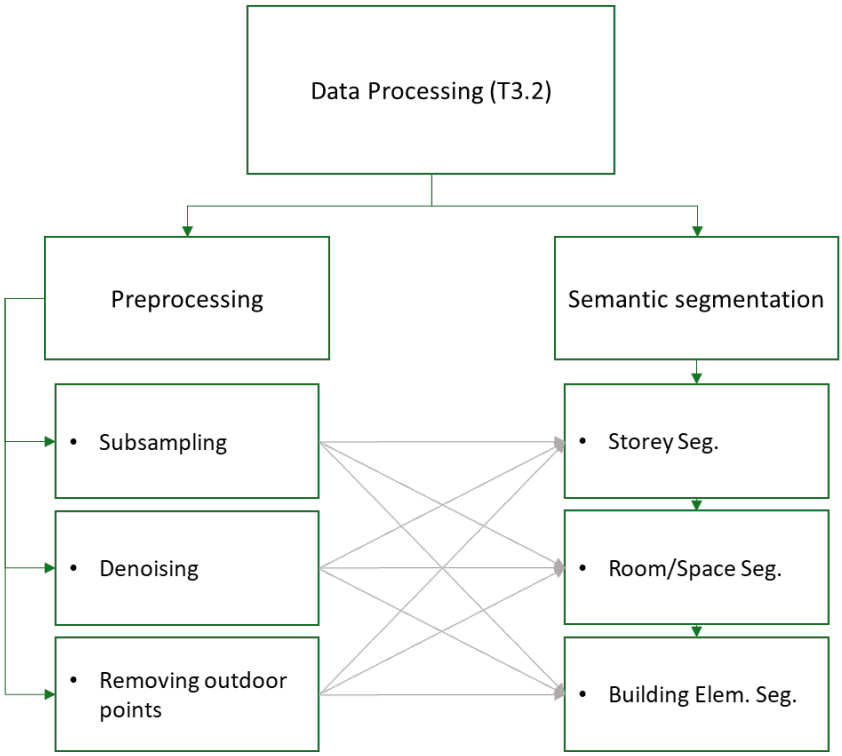

Figure 1. General Structure of the code developed in T3.2.

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.py** that segment and analyse a point cloud corresponding to a building in Bilbao. Through this process, various geospatial features are extracted and processed from the point cloud. These main needs the initial point cloud and its trajectory (see Section 6. Data Folder), the optimal parameters to process this case study (see Section 4. Parameters Folder), and their corresponding provided in *results* folder (see Section 7. Results Folder). Additionally, this folder includes another program designed to improve or refine certain elements after the **main.py** process has been completed.

4. Parameters Folder

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

5. SRC Folder

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

1. *elements_seg* is composed of six functions which correspond to the detection of building element type such as doors, walls, ceiling, and floor.
    - **doors_with_trajectory.py**: door detection process.
    - **extract_floor_ceiling.py**: ceiling and floor detection process.
    - **walls_index.py**: walls detection process.
    - **stairs.py**: stairs detection process.
    - **walls_index.py**: walls detection process.
    - **windows.py**: windows detection process.
2. *info_elements* is composed by twelve functions for the identification and analysis of the adjacency and connectivity of the point cloud elements.
    - **adjacency_intra.py**: identifies adjacent walls within the same room
    - **adjacency_walls.py**: identifies adjacent walls between different rooms
    - **graph.py**: create the graphs with NetworkX
    - **graphs.py**: creates graphs with HTML output
    - **infoBIM.py**: saves information for the IFC model of walls, floor and ceiling
    - **infoBIM_doors_windows.py**: saves information for the IFC model for doors and windows
    - **intersection_points_index.py**: identify the intersection points of the walls
    - **mod_lines.py**: removes false walls and creates a fourth wall if the algorithm has only detected three
    - **overlaps_walls.py**: if the interior walls overlap each other, it modifies them
    - **points_boundary_floor_ceiling.py**: identifies the contour points of the floor and ceiling
    - **points_global_index.py**: adjust the walls so that they form 90-degree angles
    - **points_room_order.py**: arrange the points clockwise
3. *improve_ifc* module consists of three functions designed to analyse the entire building and enhance the detection of windows, doors, and ceilings, as well as to extract information necessary for reconstructing stairs.
    - **improve_ceiling_better_ifc.py:** refines the detected floor and ceiling boundaries by filtering points near the contour of the ceiling or floor polygons.
    - **windows_better_ifc.py**: it determines the orientation of rectangular elements, computes centroids and distances, groups aligned objects, detects missing elements, and checks if windows are located within wall boundaries using plane and bounding box calculations.

- **stairs_to_IFC.py**: extracts and reconstructs stair geometry from point cloud data using plane segmentation and clustering, generating coordinates for IFC reconstruction.

4. *morph_seg* contains eight functions that constitute the morphological segmentation of the building's floors.
   - **class_empty_occupied.py**: classifies voxels in a point cloud as empty or occupied, and classifies empty voxels as interior or exterior, using a boundary polygon of the ceiling
   - **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.
   - **mix_room.py**: the room number that is poorly divided is selected and the parameters are changed so that it is divided well.
   - **occupied_voxels_classification.py**: occupied voxels are labelled regarding proximity of labelled empty voxel.
   - **pick_in_pc.py**: allows you to manually select a point from the room cloud that has been incorrectly segmented and identify which room it is in.
   - **point_cloud_classification.py**: reclassifies voxels in the point cloud.
5. *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.
6. *storey_seg* is made up of functions that will divide the point cloud in storeys, being able to use the trajectory or not.
   - **histogram_scott.py**: segment the point cloud by storeys according to the histogram using Scott's rule.
   - **segmentation_trajectory.py**: segment the point cloud by storeys using the trajectory of the building.
7. *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.
   - **convert_numpy_to_list.py**: converts all numpy.ndarray objects in a dictionary or list.
   - **functions_geom.py**: functions for geometric calculations.
   - **functions_walls.py**: functions to detect walls.
   - **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.
   - **remove_no_continuous.py**: remove non continuous celling.
   - **remove_no_continuous_floor.py**: remove non continuous floor.

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

6. Data Folder

This folder contains the input data of one case study which corresponds to five storeys of a building in Sestao (Bilbao) and its trajectory (Figure 2).

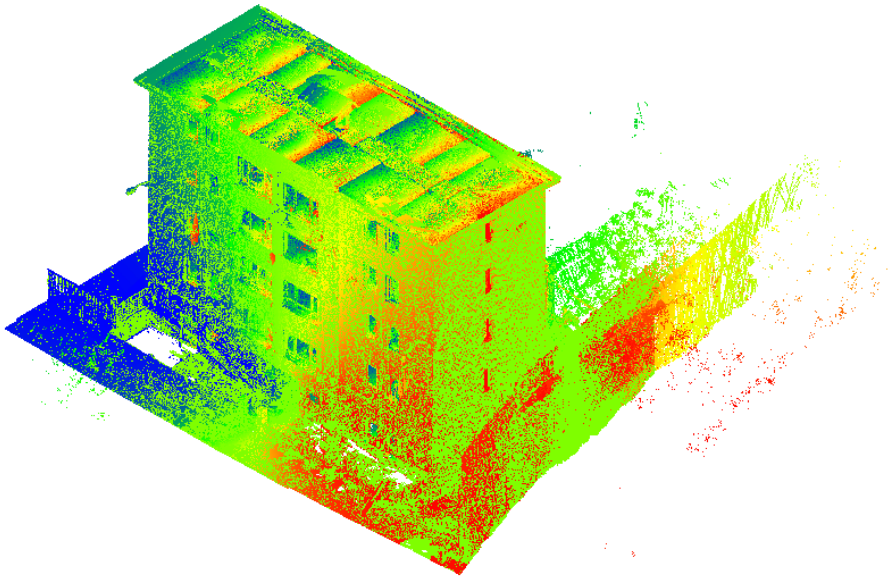

Figure 2. Input data of Case study of Bilbao.

7. Results Folder

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

- A folder with segmented storeys.
- A folder on each storey where the graphs and the segmented point cloud appear along with a folder called elements that contains information about the detected elements.

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)

JSON (https://docs.python.org/es/3/library/json.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.spatial.distance (https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html)

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

Shapely.geometry.Linestring (https://shapely.readthedocs.io/en/stable/reference/shapely.LineString.html)

Shapely.geometry.Point (https://shapely.readthedocs.io/en/stable/reference/shapely.Point.html)

Scipy.stats.mode (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mode.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.
- `path_building (str)`: path of the point cloud file.
- `path_trajectory (str)`: path of the trajectory file.
- `folder_clean_storey (str)`: path of the folder where the point clouds are located after removing outdoor points.

The code has been prepared for the creation of folders where the results of each storey will be saved, including these point cloud results, and the graphs associated with them.

9.1. Case Study Bilbao

9.1.1. Parameters

```
nt = 0.9
ransac_th_2 = 0.25
init_n = 3
itera_2 = 1000
h_2 = 0.75
g_res = 100
threshold_size = 10
voxel_size = 0.05
eps_1 = 0.1
eps_2 = 0.5
pt = 2
nt = 0.9
alpha = 3.2
ransac_th = 0.4
zmax = 1000
zmin = -1000
dist = 0.1
s_p = o3d.geometry.KDTreeSearchParamKNN(40)
pcd_in = None
vox_idx = np.array([])
clean_idx = np.array([])
vox_labels = pd.DataFrame()
empty_in_lbl = 1
occ_lbl = 10
width_door = 1
min_ratio = 0.05
threshold = 0.09
iterat = 3000
init_n = 3
h = 1.6
max_distance = 0.4
max_distancia = 0.5
```

```
min_points_to_save = 3
dist_pl = 0.5
min_angle = 0
max_angle = 40
dist_centroids_1 = 0.3
b_f = 0.5
min_adjacent_pairs = 6
max_buffer_radius = 2
dist_centroids_2 = 4
ex_v = [None]
min_p_int = 4
max_dist_int = 8
min_angle_d = 80
max_angle_d = 100
perfect_angle = 90
flat_angle = 180
angle_points_global = 45
ang_1 = 45
ang_2 = 135
images = True
write = False
dist_p = 1
distance_th = 0.2
dist_bound = 0.5
alph_boundary = 2
dist_th=0.35
min_dist_w = 0.3
um=0.005
um_d=0.1
dist_traj = 0.5
b_traj=0.05
th_d = 0.1
dist_th_w = 0.2
dist_max = 0.05
iter_i = 1
min_a = 0.5
```

```
max_a = 6
th_boundary = 0.2
th_g = 0.5
th_w = 0.1
s_h = 0.7
s_w = 0.4
w_width = 0.35
w_high = 2.2
w_th = 0.4
h_f_w = 0.4
eps_s = 0.5
pts_s = 90
ransac_s = 0.1
iterat_s = 5000
def str_el(n_storey):
    if n_storey == 0:
        se = np.ones((17,17,17))
        # se=np.ones((20,20,20))
    elif n_storey == 1:
        se = np.ones((17,17,17))
        # se=np.ones((18,18,18))
        # se=np.ones((19,19,19))
    elif n_storey == 2:
        se = np.ones((14,14,14))
        # se=np.ones((16,16,16))
    elif n_storey == 3:
        se = np.ones((15,15,15))
        # se=np.ones((18,18,18))
    elif n_storey == 4:
        se = np.ones((14,14,14))
        # se=np.ones((18,18,18))

    return se
```

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 = "\\csBilbao"
folder_results = "C:\\...\\Code\\results"
path_building = path + folder_cloud + "\\Bilbao_all_AutoCleaned.txt"
path_trajectory = path + folder_cloud + "\\Trajectory.txt"
folder_clean_storey = folder_results + folder_cloud + "\\Clean_Storeys"
```

In this case, the point cloud is in txt format whose attributes are x, y, z (Figure 3). The process starts by applying storey segmentation using the trajectory of the building.

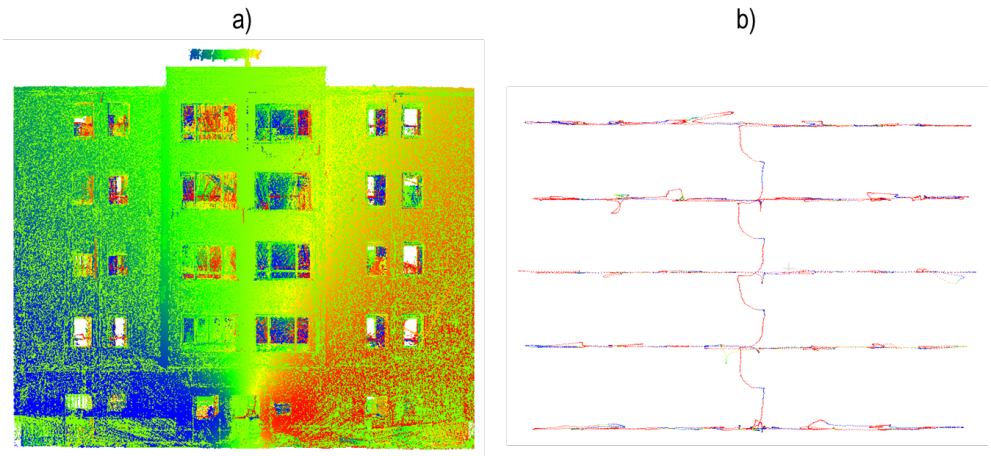

Figure 3. a) Initial point cloud cleaned, b) trajectory of point cloud

The gaps between the different storeys are identified, and the point cloud is segmented. The result is shown in Figure 4.

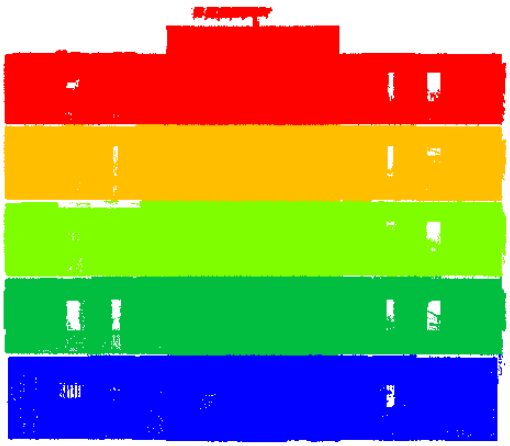

Figure 4. Result of storey segmentation

Once the point cloud has been segmented by storeys, the room segmentation process begins in each of them. First, the stairwell area is extracted to make the mathematical morphology process more precise.

For the room segmentation process, we begin with the voxelization of the point cloud, removing noise and identifying the points corresponding to the floor and ceiling. The contour is then identified from the ceiling point cloud, and the voxels are classified as indoor and outdoor. Indoor voxels are subdivided into occupied (when they contain points) or empty (when they do not contain points). 3D morphological operations are then applied, as shown in Figure 5.

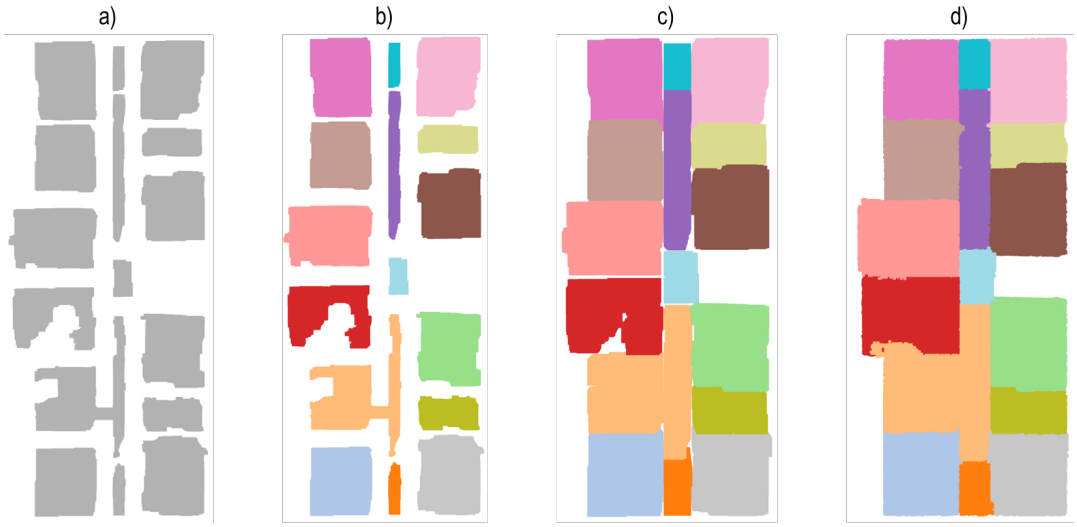

Figure 5. a) Erosion, b) Individualization, c) Dilation, d) Classification

In some cases, a room needs to be reprocessed because it was not segmented correctly as shown in Figure 6.

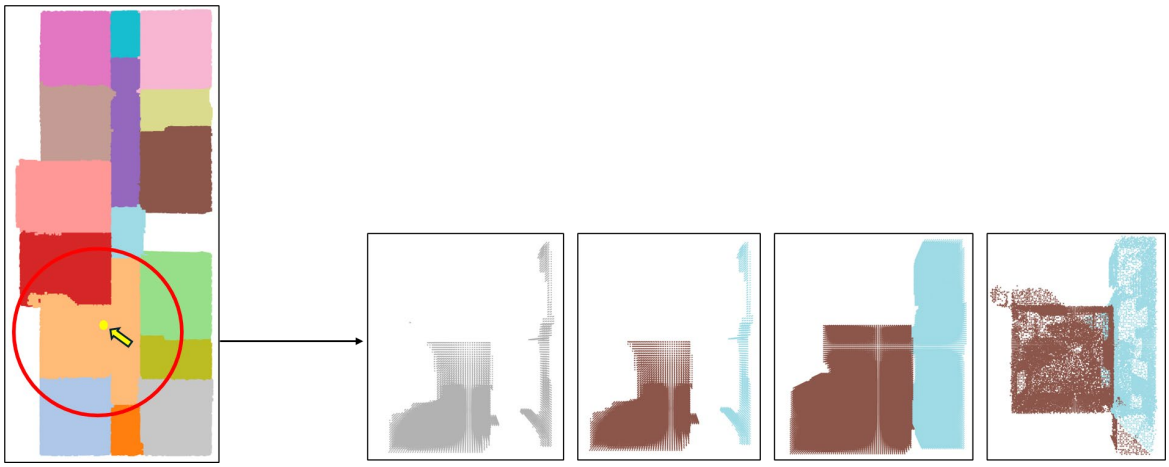

Figure 6. Reprocessing a room that has not been segmented

The result of this process is a point cloud segmented by room (Figure 7).

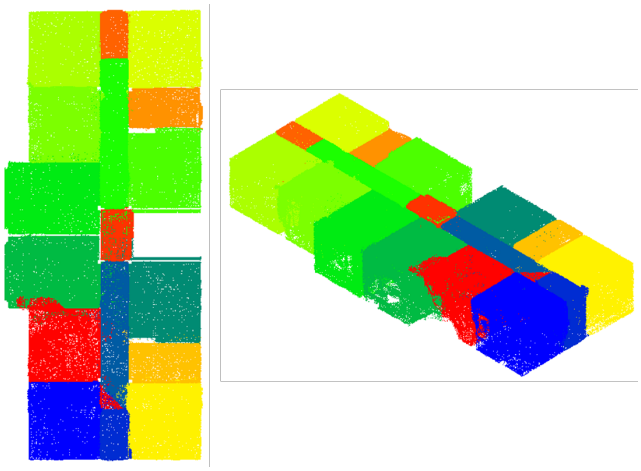

Figure 7. Result of room segmentation

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 8 follow the scheme shown in Section 10.

	x	y	z	id_storey	id_room	id_element
0	-8.489944	3.829241	-4.452875	3	4	2
1	-8.257915	3.831081	-4.258875	3	4	2
2	-8.087894	3.832361	-4.415875	3	4	2
3	-8.066825	3.829891	-4.230875	3	4	2
4	-6.888545	3.833561	-4.288875	3	4	2
...	...	...	...	...	...	...
3366188	-12.397346	-1.723912	-1.855875	3	17	2
3366189	-12.551295	-1.727792	-1.751875	3	17	2
3366190	-12.544146	-1.733622	-1.800875	3	17	2
3366191	-12.262115	-1.731512	-1.717875	3	17	2
3366192	-12.290735	-1.667212	-1.564875	3	17	1

[3366193 rows x 6 columns]

Figure 8. Point cloud Data Frame format

If the data is displayed depending on the attribute, the results shown in Figure 9 can be seen.

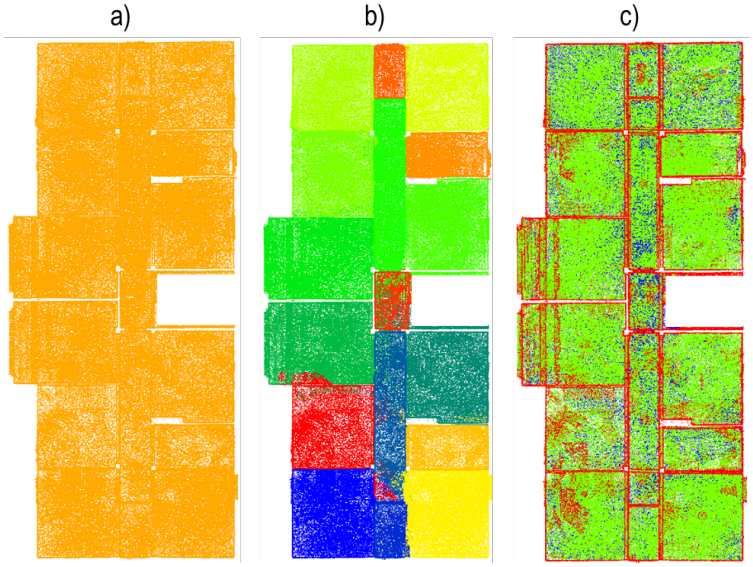

Figure 9. Data visualizated based on a) id_storey attribute, b) id_room attribute, c) id_element attribute.

The walls of the different rooms shown in Figure 10.

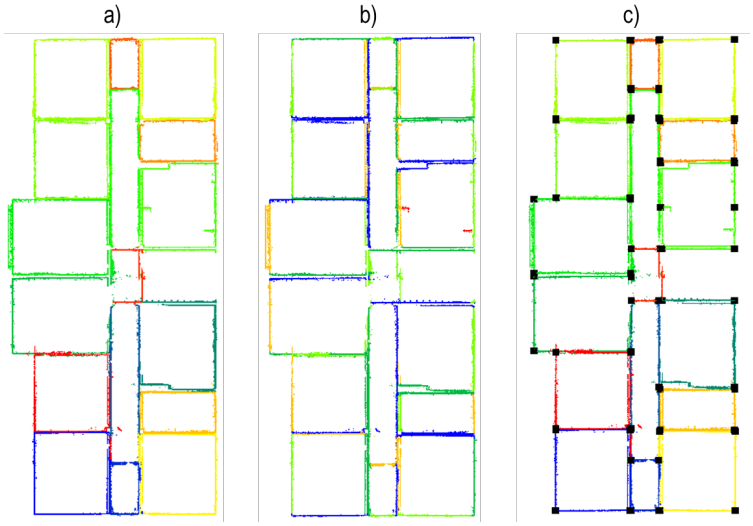

Figure 10. a) Walls by room, b) each wall in each room, c) intersection points of walls

Figure 11 show intersection in the floor and in the ceiling.

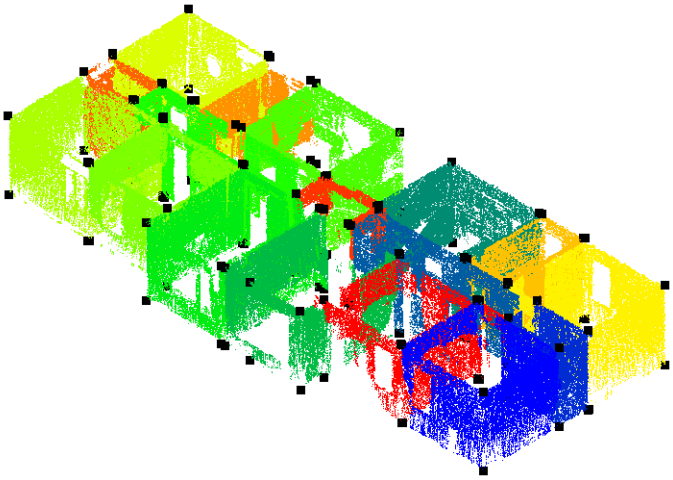

Figure 11. Points intersection in the floor and in the ceiling.

Once the walls have been identified (including the identification of exterior and interior walls), the adjacency relationships between the walls of different rooms, as well as between the walls of the same room, are determined. As a result, the adjacency between the rooms is established. To represent these relationships, graphs are used, where the main nodes correspond to the rooms, the sub nodes represent the walls, and the edges indicate the adjacency relationship between them.

In this case, the graphs are called inter-room adjacency graphs and intra-room adjacency graphs. These graphs represent the adjacency relationships between rooms (Figure 12.b) and the adjacency relationship between rooms across walls (Figure 12.a), while the latter models the adjacency relationships within the same rooms, across walls (Figure 12.c).

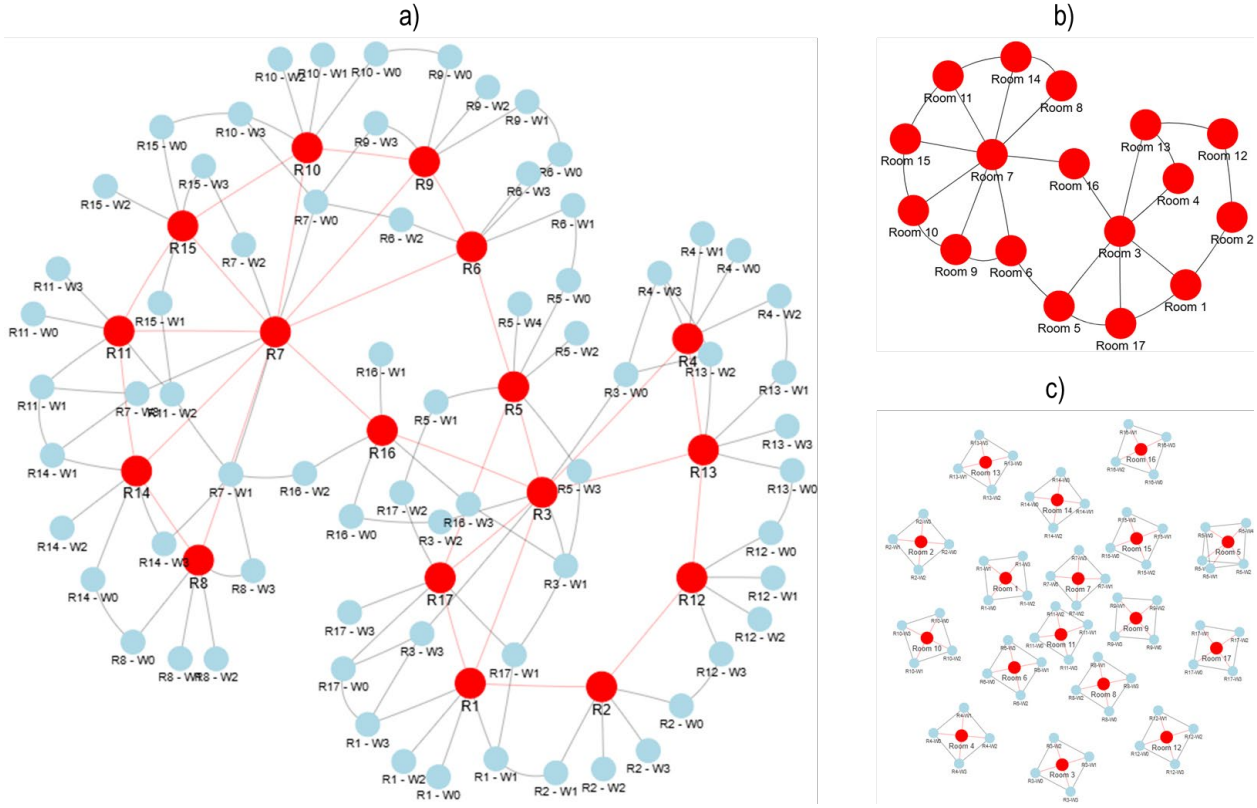

Figure 12. a) Inter-room adjacency graph with walls, b) inter-room adjacency graph, c) intra-room adjacency graph

Door detection is then performed, which involves analyzing both the floor plan trajectory and its room segmentation (Figure 13.a). Since the points along the trajectory are continuous, the zones where a room change occurs are identified. Then, using information from previously detected interior walls, the precise location of the doors is calculated. The four endpoints of each door are determined using standard-based preset values (Figure 13.c). Furthermore, to reduce process time, the trajectory has been segmented every half meter (Figure 13.b).

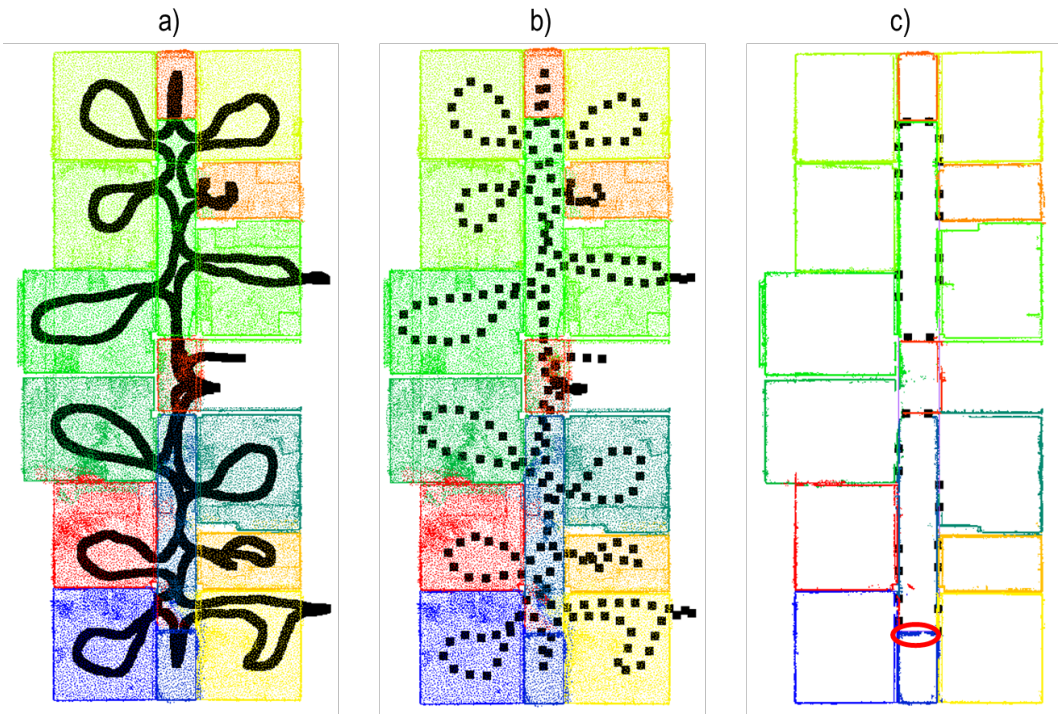

Figure 13. a) Room segmentation with trajectory, b) room segmentation with trajectory segmented, c) door points obtained

In addition, the graph called inter-room connectivity graph is also obtained as can be seen in Figure 14.

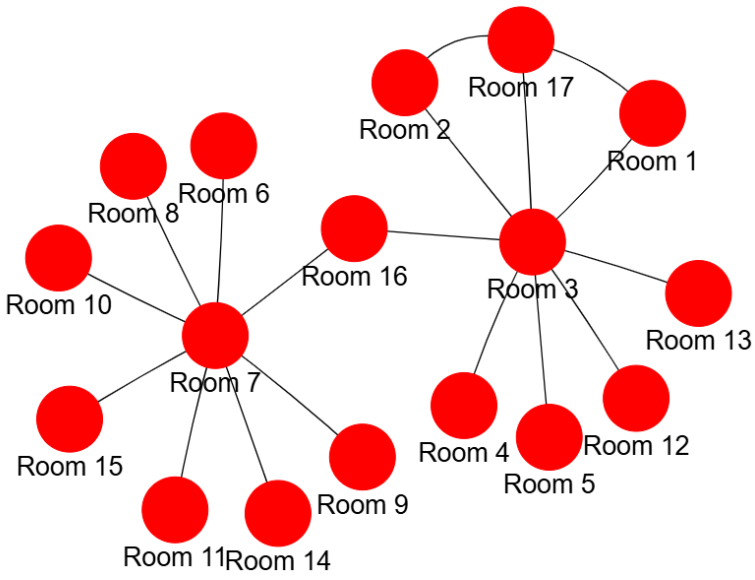

Figure 14. Inter-room connectivity graph

Finally, the windows are detected. To do this, the exterior walls that match the contour of the floor plan's point cloud are selected and projected in 2D to visualize the empty areas where the windows should be located. The center is then identified, and the four end points of each window are generated using preset values. These points are then re-identified in the 3D point cloud as can be seen in Figure 15. Window openings in point clouds are detected if there are no occlusions caused by objects, such as furniture.

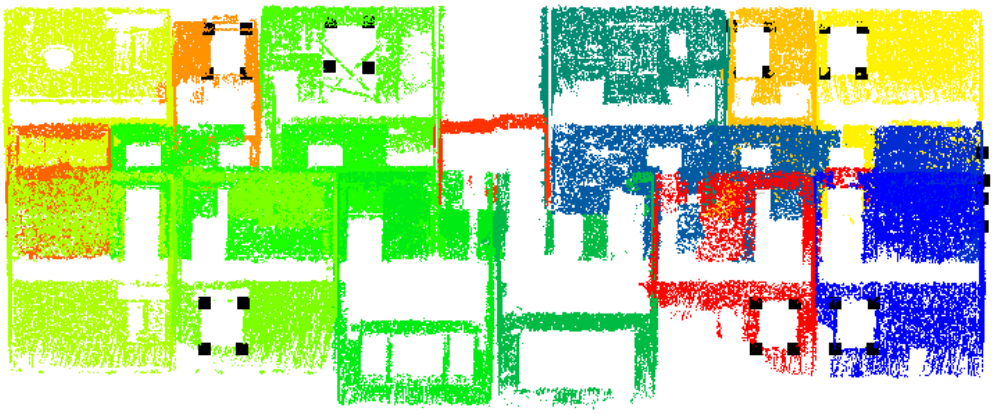

Figure 15. Window point detection

To develop the final IFC model, all the information collected throughout the process was recorded. This information includes the points that define the floor outline, the roof outline, exterior walls, interior walls, doors, windows, and stairs, as illustrated in Figure 16. This text file contains the following elements:

- storey_id: Storey number.
- element_type: Element type, such as floor, ceiling, wall, door, window, or staircase.
- element_id: Unique identifier of the element.
- is_external: Indicates whether the element is internal or external. This is crucial for differentiating between exterior and interior walls.
- height: Height of elements that require it, such as walls or ceilings.
- opening_in_wall: Specifies the number of the wall in which the doors or windows are inserted.
- np_pts: Number of points that define the element's geometry.

The following lists $pti\_x, pti\_y, pti\_z\ points\ (with\ \ i \in \{1, \dots, n\}, n\ is\ the\ last\ point\ of\ the\ element)$ that delimit each element, arranged in order.

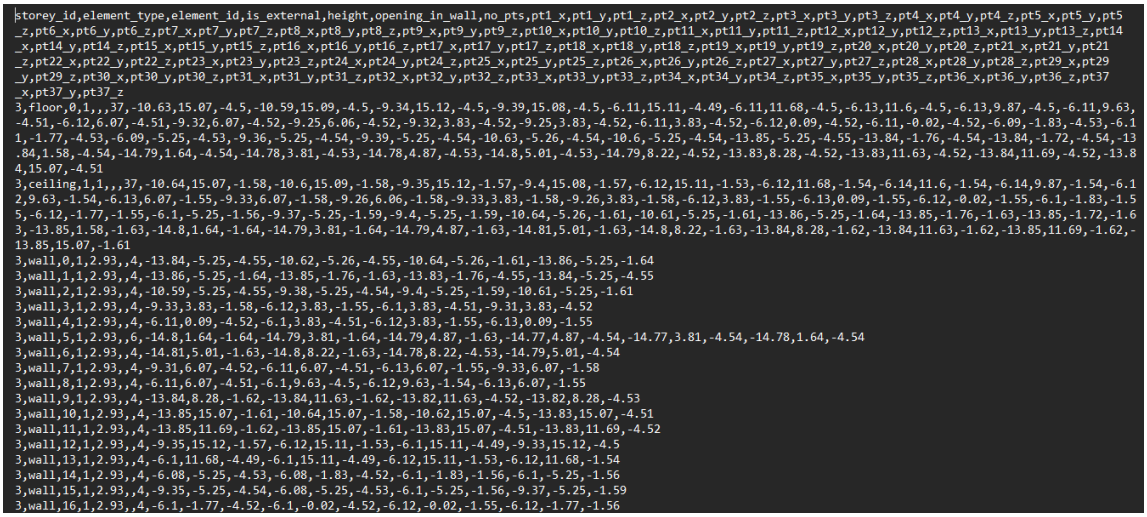

Figure 16. Format of the saved data

Finally, the creation of the IFC model corresponding to the processed storey is shown in Figure 17.

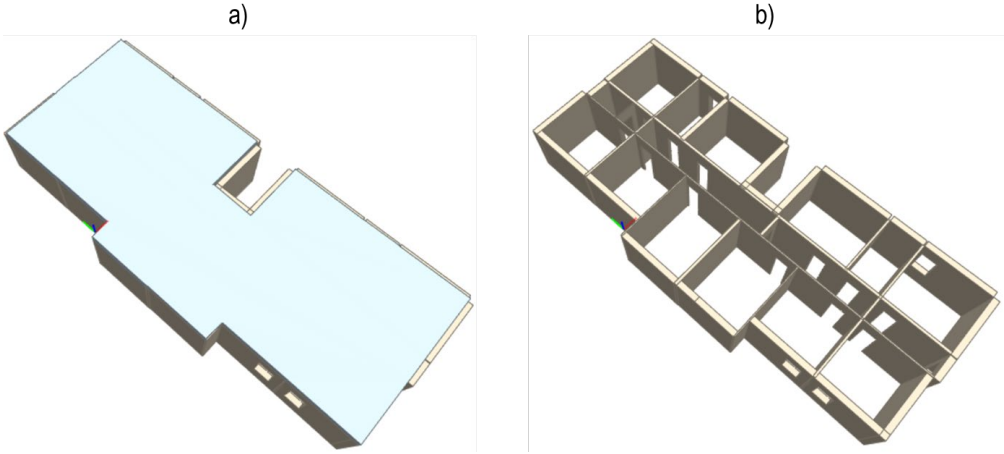

Figure 17. IFC model of one storey

If the process is repeated for all storeys of the building, the IFC model in Figure 18 would be obtained.

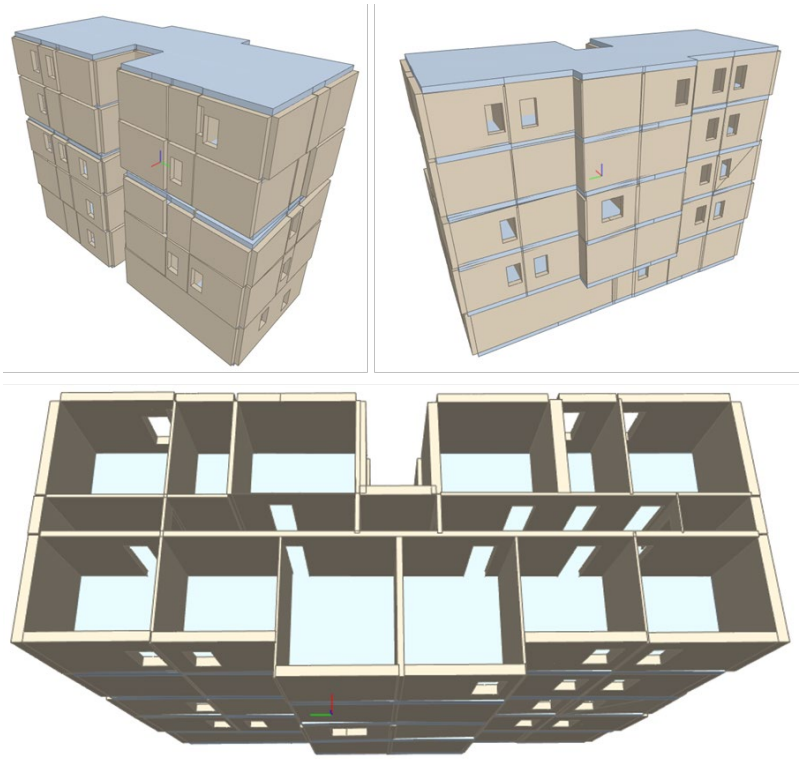

Figure 18. Final IFC Model

Once the main process is completed, the **improved_ifc.py** program runs, producing the enhanced results for windows (Figures 19.a and 19.b) and for the floors and ceilings (Figure 19.c), which contribute to the final model shown in Figure 20.

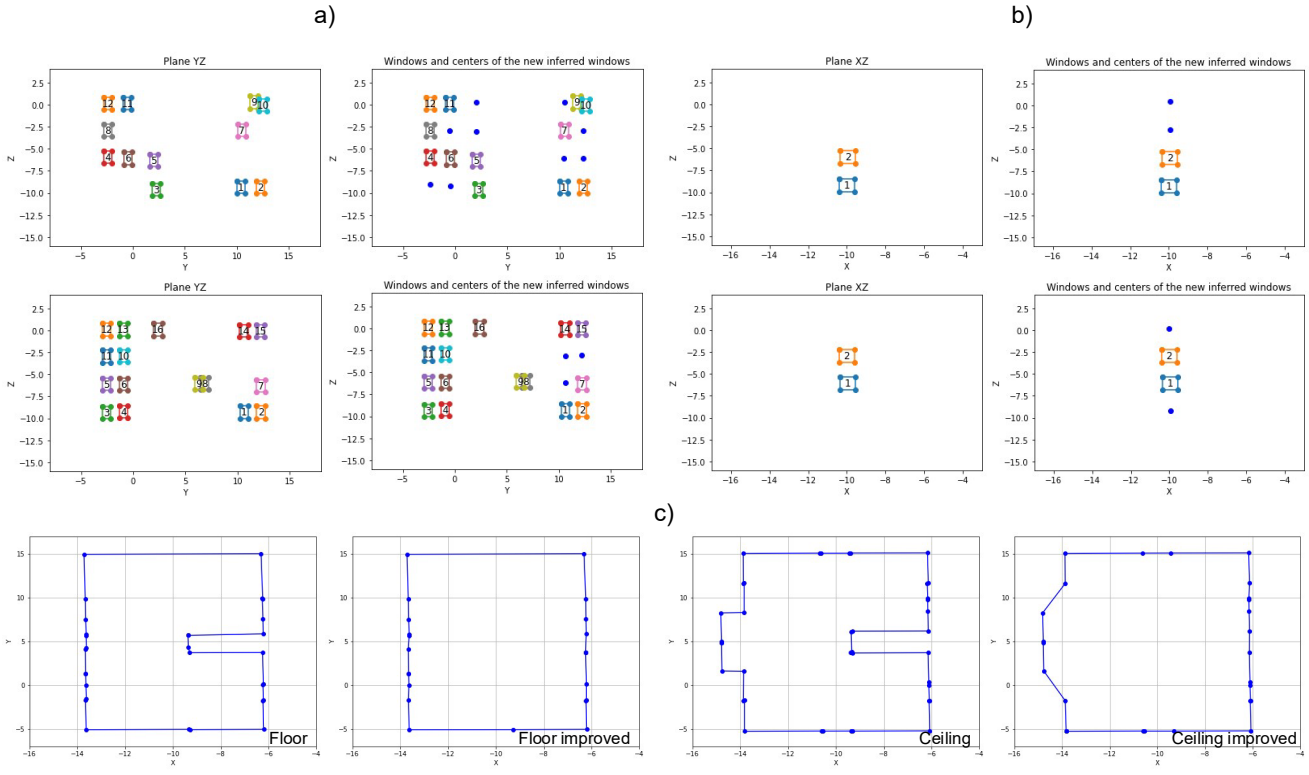

Figure 199. Improvements on the IFC Model: a) Frontal and rear façade, c) Lateral façades, and c) Floor and ceiling of storey 0

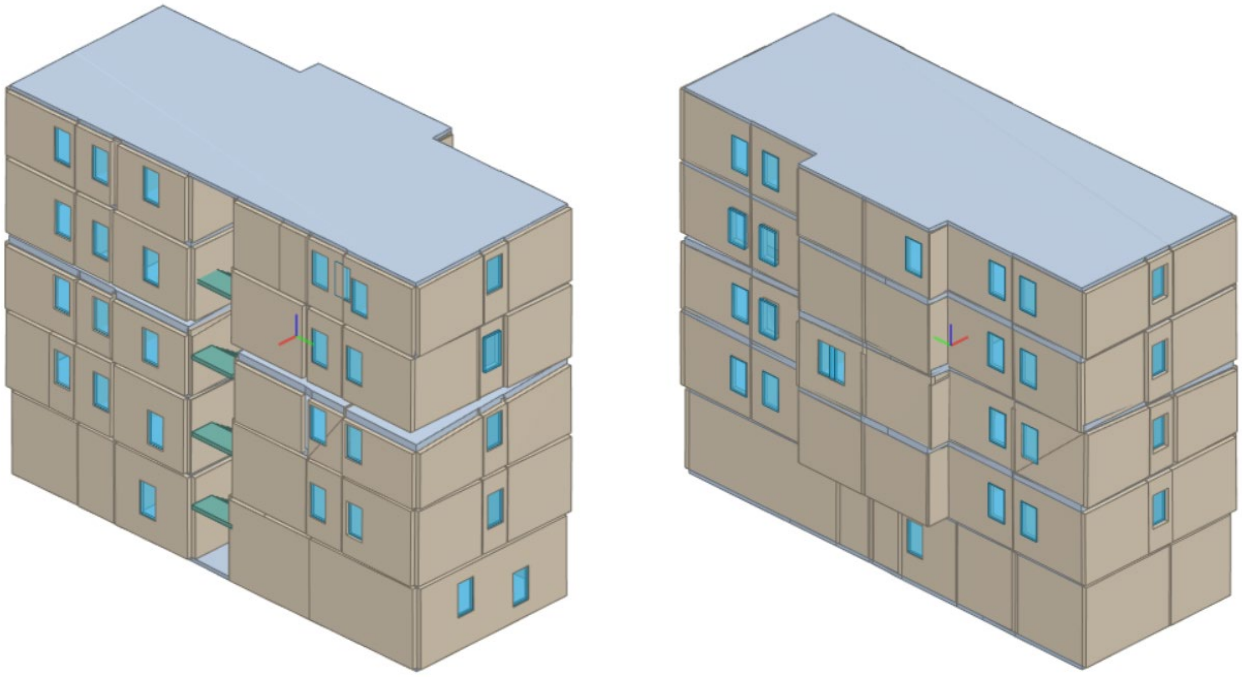

Figure 20. Improved IFC Model

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 19) need to be changed by Scalar (Figure 20).

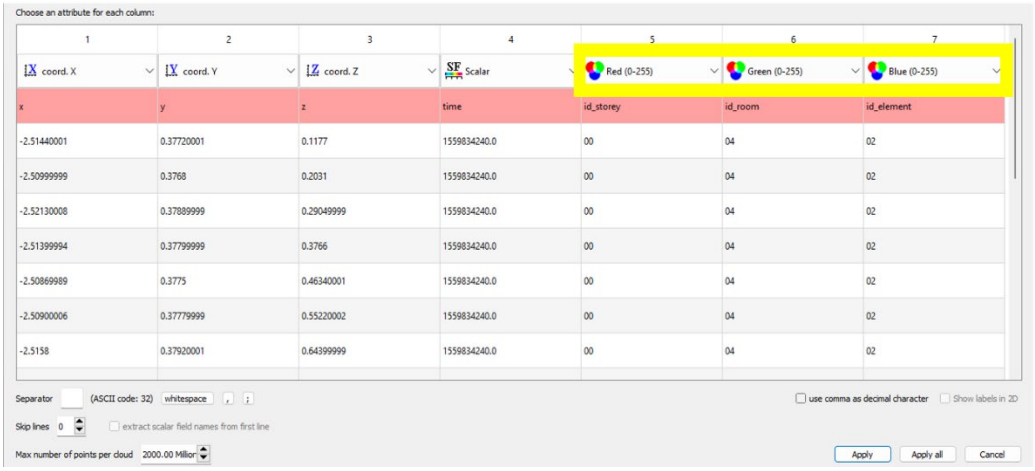

Figure 20. Red, Green, and Blue default attributes.

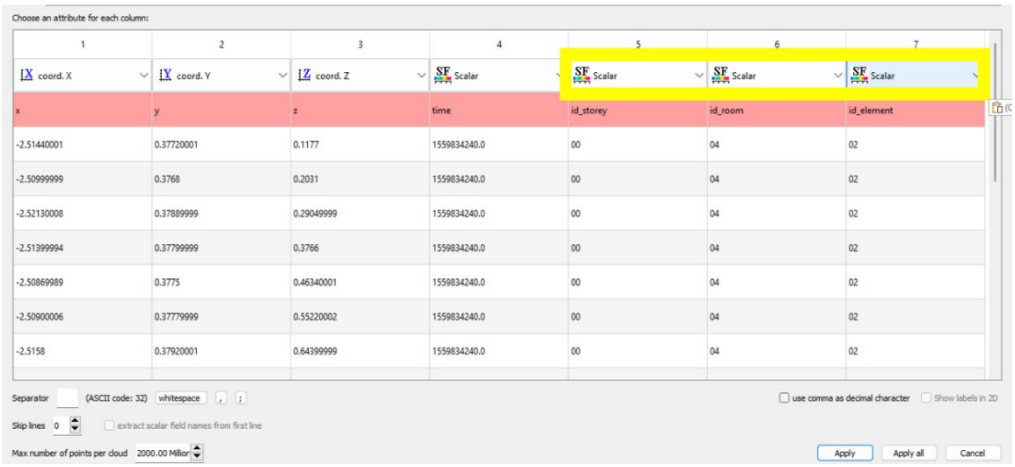

Figure 21. Scalar attributes.

- To execute the room/space segmentation process, the data frame of point cloud must have the *id_storey* attribute.
- Attributes:

ATTRIBUTES: Residential Building

- x, y, z, time, … ⟶ Input variables of the point cloud
- id_storey ⟶ Indicate the storey number
- id_room ⟶ Indicate the room number
- id_element ⟶ Indicate the element type (see table)
- id_entity ⟶ Indicate the wall number (99 indicates that it is not a wall)

Element	Label
Floor	0
Ceiling	1
Walls	2
Columns	3
Beams	4
Stairs	5
Others	99

Figure 22. 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, corresponding with the residential pilot. 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.