Deliverable Title

Document type	**Deliverable D3.2.** Computational algorithms developed in T3.2.
Document Version / Status	**1.0 - 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

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. 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*, and *door*. However, the code is still under continuous improvement, and other element types such as windows and columns will be considered. Figure 1 shows a schema of the code general structure.

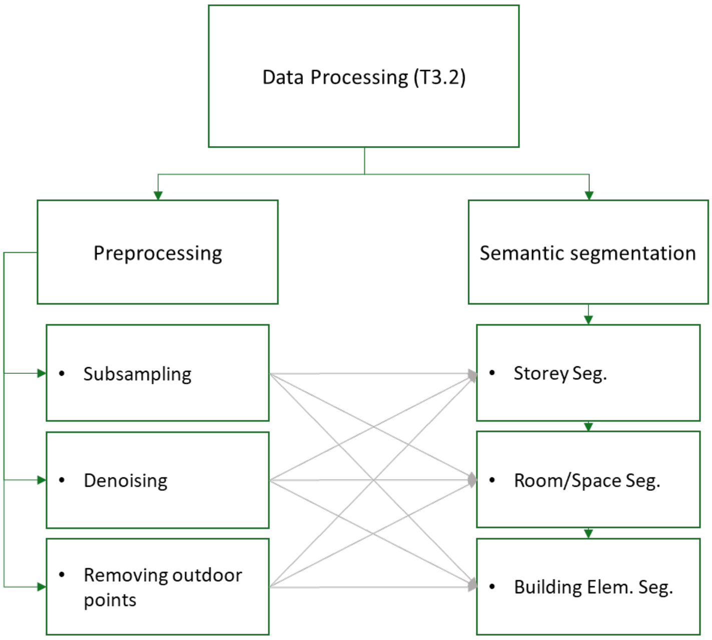


Figure 1. General Structure of the code developed in T3.2.

2. Structure of the General Folder

According to the RecycleBIM proposal, computational algorithms developed in T3.2. are uploaded to GitHub for global sharing. The folder containing the code is structured into four subfolders: *test* (main program), *data* (input data), *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/B9NtDcMaHnzHoTx. Figure 2 shows the general structure of the code folder, and more details are provided in the following sections.

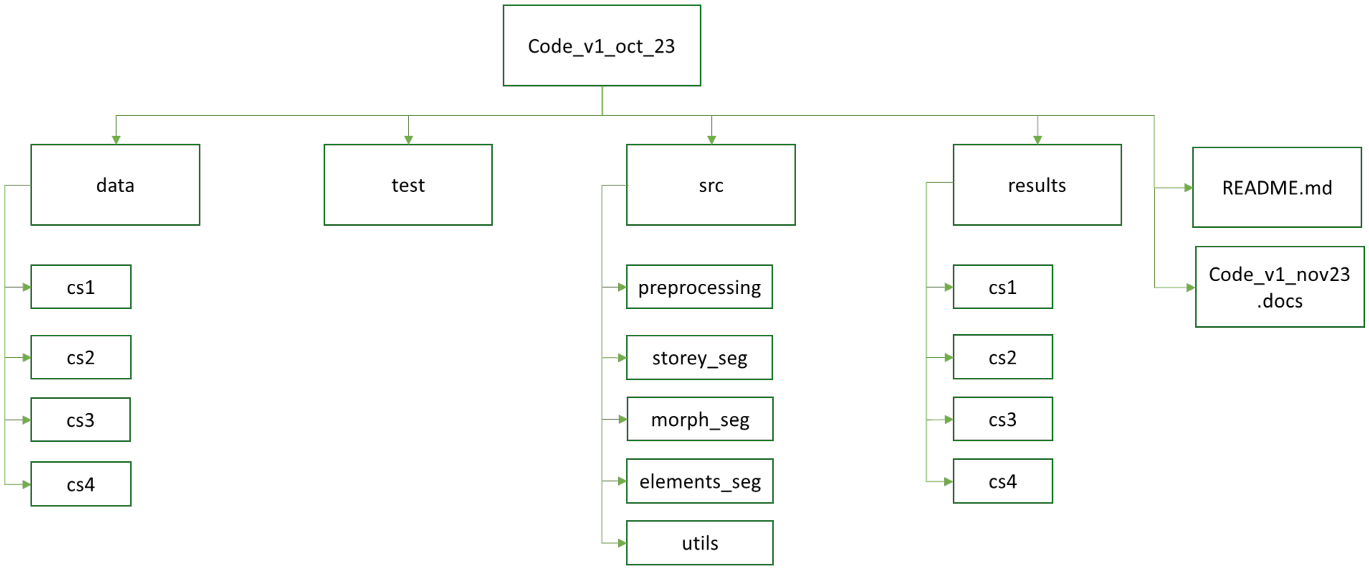


Figure 2. General structure of the folder

3. Test folder

Although just one main program is conceived, for the sake of clarity this folder includes four main programs corresponding to each of the four case studies provided in *data*. Each main program includes the optimal parameters to process each case study, and results correspond to the ones provided in *results* folder (see Section 5. Data Folder).

1) **main_cs_1.py** (case study 1)
2) **main_cs_2.py** (case study 2)
3) **main_cs_3.py** (case study 3)
4) **main_cs_4.py** (case study 4)

4. SRC Folder

This folder is organized into five subfolders: (1) *elements_seg*, (2) *morph_seg*, (3) *preprocessing*, (4) *storey_seg* and (5) *utils*.

1. *elements_seg* is composed of three functions which correspond to the detection of building element type such as doors, walls, ceiling, and floor.
   - **doors.py**: door detection process.
   - **extract_floor_ceiling.py**: ceiling and floor detection process.
   - **walls.py**: walls detection process.
2. *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.
3. *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.
4. *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.
    - **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.
5. *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.
    - **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.

5. Data Folder

This folder contains the input data of four case studies:

1) *cs1* (Case Study 1): point cloud corresponding to a storey of the School of Mining and Energy Engineering at the University of Vigo (Figure 3).

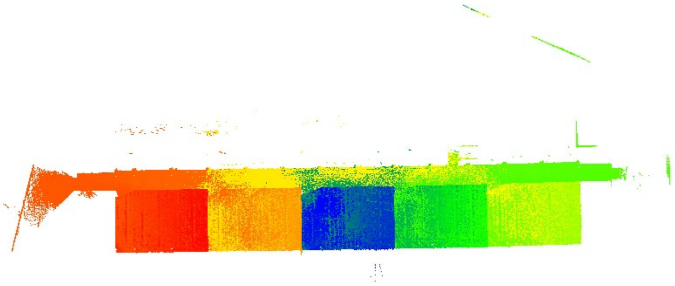


Figure 3. Input data of Case study 1.

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

2) *cs2* (Case Study 2): point cloud and trajectory provided by the ISPRS Benchmark on Indoor Modelling[2] (Figure 4).

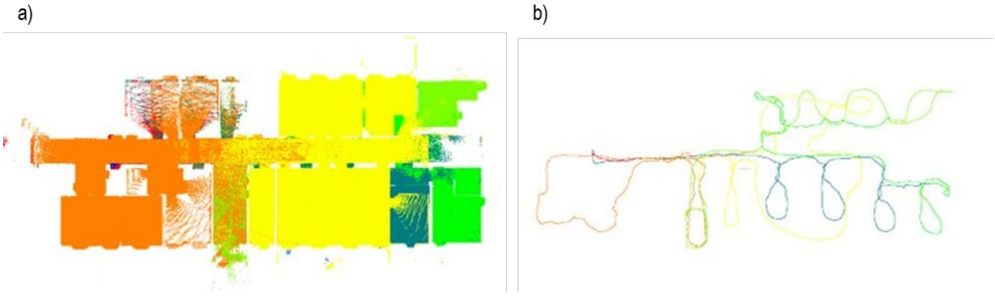


Figure 4. Input data of Case study 2. a) Point cloud, b) trajectory.

3) *cs3* (Case Study 3): point cloud corresponding to the 22nd and 23rd storeys of a building to-be-demolished in Lugo (Spain) (Figure 5).

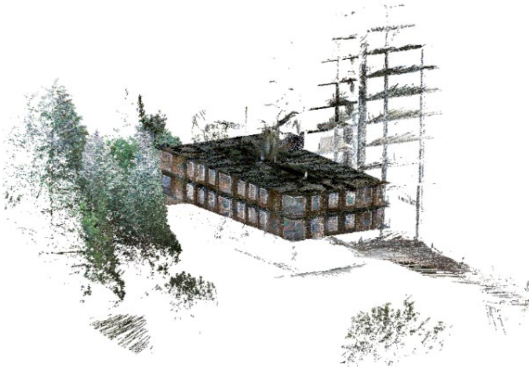


Figure 5. Input data of Case study 3.

4) *cs4* (Case Study 4): point cloud corresponding to the 37th and 38th storeys of the same building as cs3 (Figure 6).

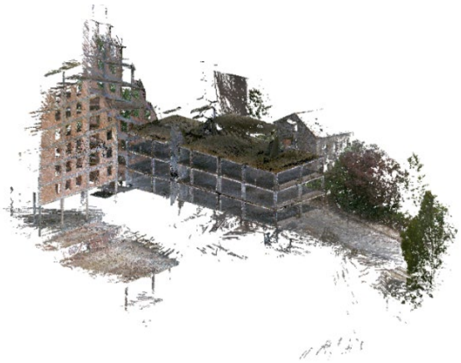


Figure 6. Input data of Case study 4.

[2] https://www.isprs.org/education/benchmarks/IndoorModeling/Default.aspx

6. Results Folder

Results contain the outputs of processing the input data using the parameters included in the corresponding main programs (*test* folder). In each subfolder (cs1, cs2, cs3, cs4), 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'.
- If the point cloud has more than one storey, the result will be several files in txt format (one for each floor) with the same structure as the previous case: previous attributes (i.e. x, y, z, r, g, b, i, etc.) and new attributes such as 'id_storey', 'id_room', 'id_element'.

7. 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)

8. Parameters

This section includes a list of the parameters involved in the code and can be modified by the user. More information is available in the docstring of each function.

- `dss (float):` subsampling distance in meters of the trajectory (i.e., 0.1).
- `pcd_bin (float):` bin size of point cloud in meters (i.e., 0.1).
- `traj_bin (float):` bin size of trajectory in meters (i.e., 0.3).
- `n_max (int):` number of peaks (i.e., 2).
- `min_hor_dist (float):` minimum distance between two peaks in meters (i.e., 1).
- `n_bins (int):` number of bins of point cloud histogram (i.e., 33).
- `dist_b (float):` distance in meters between floor and ceiling of a storey (i.e., 3).
- `voxel_size (float):` size of the voxel grid to filter the cloud in meters (i.e., 0.1).
- `eps (float):` maximum distance in meters between two samples for one to be considered as in the neighbourhood of the other (i.e., 0.6).
- `pt (int):` number of samples (or total weight) in a neighbourhood for a point to be considered as a core point. This includes the point itself (i.e., 2).
- `nt (float):` threshold used to determine which points will be selected based on the orientation of their normal in meters (i.e., 0.9).
- `alpha (float):` level of detail or complexity of the alpha shape. Larger values of alpha will produce simpler, smoother envelopes, while smaller values of alpha will produce more detailed and complex envelopes that better fit the details of the input points. Is measured in meters (i.e., 3.2).
- `ransac_th (float):` threshold used to determine whether a point is close enough to the estimated plane. If a point is a distance greater than `ransac_th` from the plane, it would be considered an outlier in the context of plane fitting. Is measured in meters (i.e., 0.12).
- `dist (float):` maximum distance in meters from a point to an estimated plane to be considered an inlier (i.e., 0.5)
- `width_door (float):` size in meters to define the structure of the element (i.e., 1.5).
- `pts_door (float):` distance in meters that you define to classify a point as a "door". Points that are at a distance less than pts_door will be considered "non-doors", and points that are at a distance equal to or greater than `pts_door` will be considered "doors" (i.e., 2).
- `eps_door (float):` maximum distance in meters between two samples for one to be considered a neighbour of the other. Basically, it determines how close two points must be to be considered part of the same cluster. If `eps_door` is very small, the algorithm may not find many clusters and group many points into the same cluster. If it is too large, the algorithm may not find distinct clusters (i.e., 2).
- `min_pt_door (float):` minimum number of points that must be inside `eps_door` for a point to be considered a "center point" of the cluster. In other words, it is the minimum number of points that must be close to a point for it to be considered part of a cluster. If the number of points within `eps_door` around a point is less than `min_pt_door`, that point will be considered a "border point" and will not be included in the cluster (i.e., 2).
- `min_ratio (float):` minimum left points ratio to end the detection. Is measured in meters (i.e., 0.05).
- `threshold (float):` maximum distance in meters that a point can have to an estimated plane to be considered an inlier (i.e., 0.09).
- `iterations (int):` maximum number of iterations to detect walls (i.e., 1000).
- `init_n (int):` number of initial points to be considered inliers in each iteration (i.e., 3).
- `h (float):` difference between the lowest and highest Z values greater than `h.` Is measured in meters (i.e. 1.6).
- `image (str):` images are shown or not (i.e., `True`).
- `write (str):` new files are saved or not (i.e., `False`).

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. For example, when a point cloud is subdivided in storeys, the output file is named in the form **x_storey_y.txt**, being `x` the name of the case study and `y` the storey to which it belongs. For example: **cs2_storey_0.txt** indicates that it belongs to case study *cs2* and to storey 0. Moreover, the execution time of each file is also provided with the format **time_seg_xxx.txt** with `xxx` being the name of the files described above.

9.1. Case Study 1

9.1.1. Parameters

```
-   voxel_size: 0.1
-   eps: 0.3
-   width door: 1.5
-   ransac_th: 0.12
-   h: 2.7
-   pts_door: 2.5
```

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 7). Furthermore, the point cloud is composed of one storey. Accordingly, `'id_storey'` is set to ‘0’, by default.

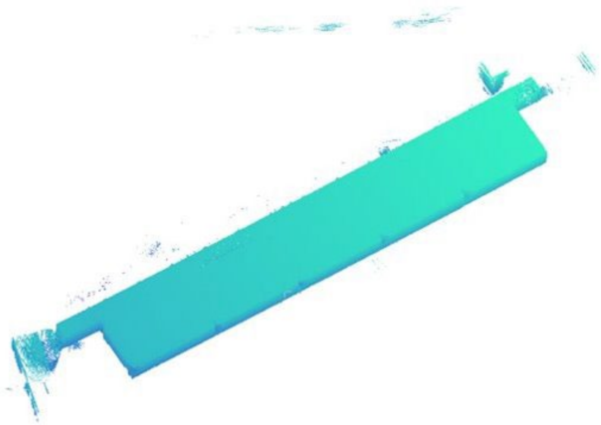
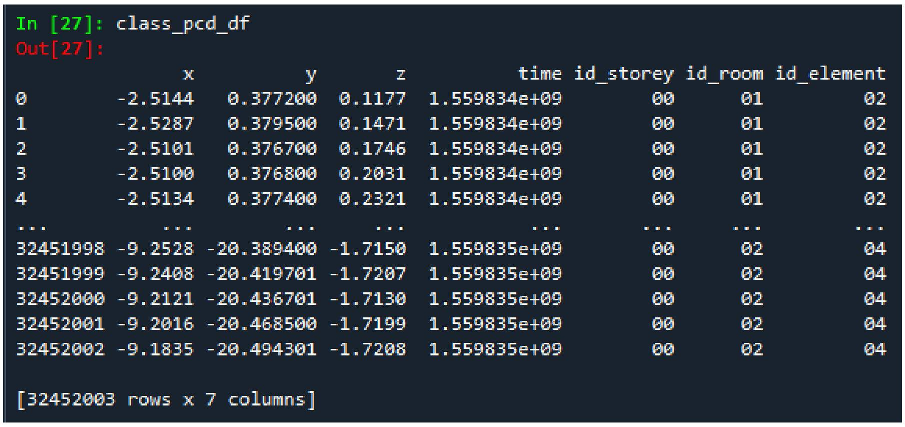


Figure 7. Initial point cloud.

Once the different segmentation functions are applied, a data frame is obtained with the format shown in Figure 20. The attributes that make up the data frame in Figure 8 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 8. Point cloud Data Frame format.

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

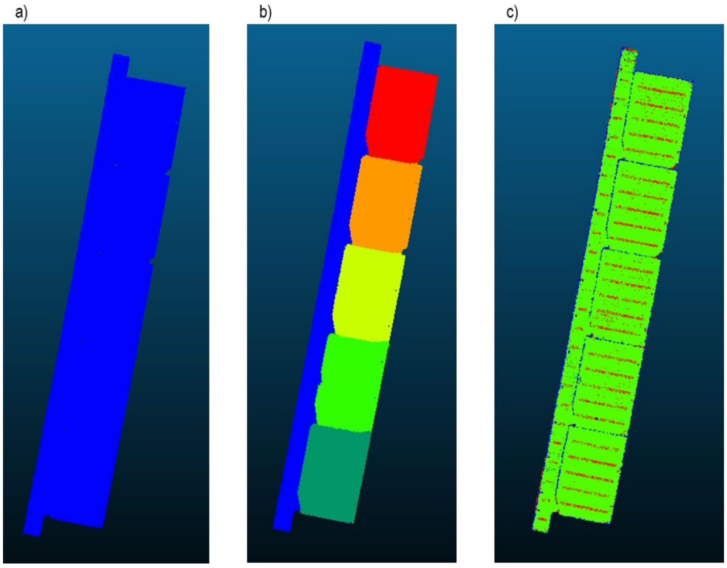


Figure 9. Data visualizated based on a) *id_storey* attribute, b*) id_room* attribute, c) *id_element* attribute.

9.2. Case Study 2

9.2.1. Parameters

Storey_0:

- `voxel_size: 0.1`
- `eps: 0.6`
- `width door: 1.2`
- `ransac_th: 0.12`
- `h: 1.6`
- `pts_door: 2`

Storey_1:

- `voxel_size: 0.1`
- `eps: 0.7`
- `width door: 1.5`
- `ransac_th: 1.5`
- `h: 1.6`
- `pts_door: 2`

9.2.2. Implementation

To initiate the processing of case study 2, the following paths need to be established:

```
path = “C:\\...\\Code\\data”
folder_cloud = “\\cs2”
name_cloud = \\Initial cs2.txt
```

```
name_trajectory = “\\Trajectory_Initial_cs2.txt”
folder_results = “C:\\...\\Code\\results”
```

Both files are in txt format and consist of the `x`, `y`, `z` and `time` attributes. Point cloud of case study 2 has two storeys, so the process starts by segmenting the point cloud into storeys by trajectory analysis.

In this case, the full names of the point cloud path, and their trajectory are:

```
pcd_cloud = path + folder_cloud + name_cloud
pcd_traj = path + folder_cloud + name_trajectory
```

Figure 10 shows the input data of case study 2, composed of point cloud and trajectory.

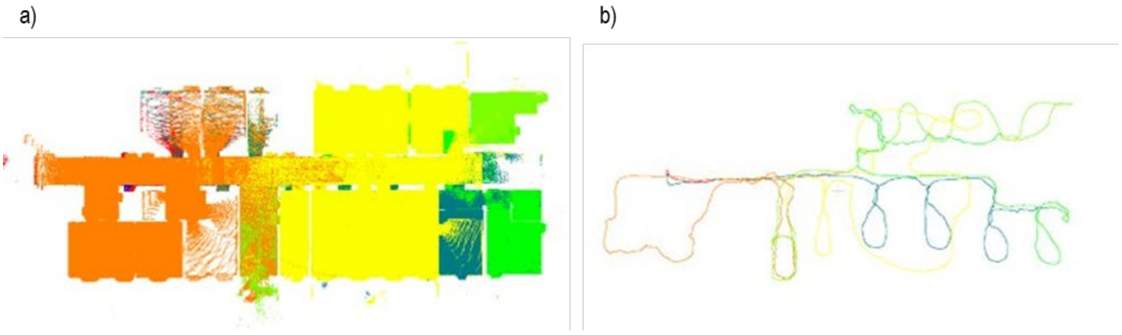
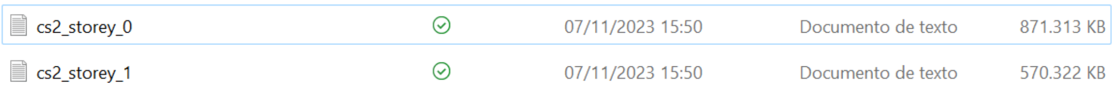


Figure 10. Input data. a) Point cloud, b) trajectory of point cloud.

The result of segmenting the point cloud into storeys with trajectory are two files (shown in Figure 11), each one belonging to a different storey in txt format.



cs2_storey_0	✓	07/11/2023 15:50	Documento de texto	871.313 KB
cs2_storey_1	✓	07/11/2023 15:50	Documento de texto	570.322 KB

Figure 11. Data frame results in txt format.

Moreover, it will be saved in the results folder that we have created, and each file will have a new column called *id_storey* (0 for the lower storey, and 1 for the upper storey). Figure 12 shows the segmented point cloud coloured by *id_storey* attribute, a) before removing outdoor points, b) after manually removing outdoor points.

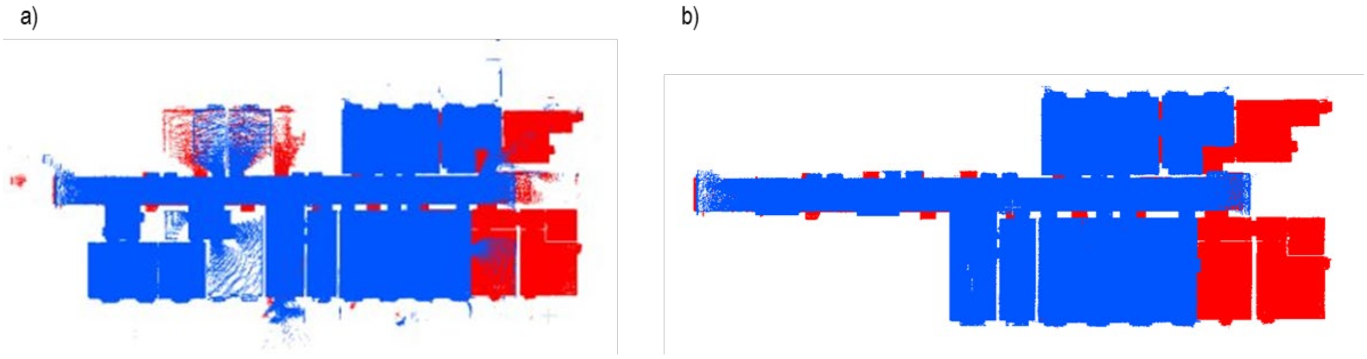


Figure 12. Point cloud a) before removing outdoor points, b) after manually removing outdoor points.

Once the point cloud is segmented by storeys, each storey is further segmented by rooms/spaces and by building element type. Because the process is the same as case study 1 and 3, it is not described to avoid redundancies.

9.3. Case Study 3

9.3.1. Parameters

```
n_bins: 33
```

Storey_0:

```
-   voxel_size: 0.1
-   eps: 0.6
-   width door: 1
-   ransac_th: 0.12
-   h: 1.6
-   pts_door: 2
```

Storey_1:

```
-   voxel_size: 0.09
-   eps: 0.6
-   width door: 0.8
-   ransac_th: 0.5
-   h: 1.6
-   pts_door: 2
```

9.3.2. Implementation

Storey Segmentation is also applied to cs3, but unlike the previous case, trajectory is not available. Subsampling with a spatial resolution of 0.05 m has been applied to the original point cloud to improve efficiency in point cloud manipulation and analysis. Figure 13 shows a) the input point cloud, b) point cloud cleaned and segmented by storeys.

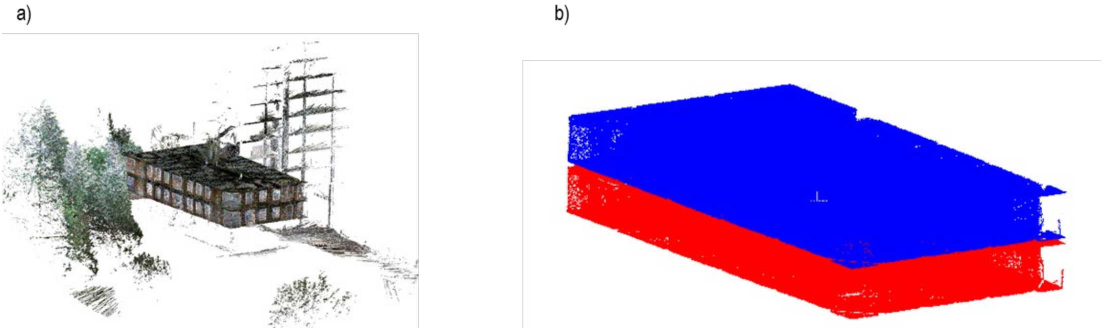


Figure 13. Point cloud a) original, b) cleaned and segmented by storeys.

Next, the process of segmentation by rooms begins on each of storeys and has as a result the one provided by Figure 14 in, for example, the lower storey.

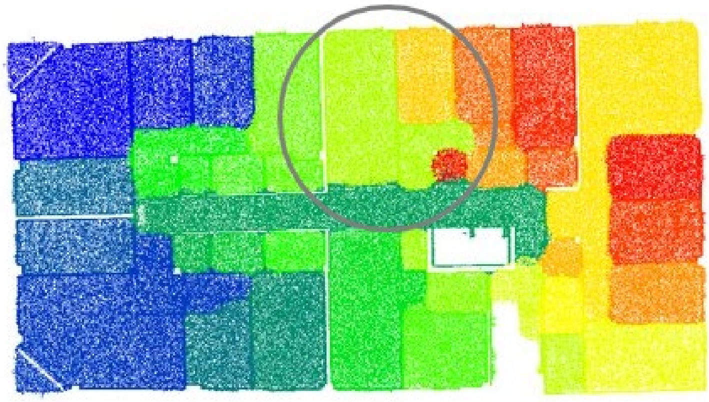


Figure 14. One room is undersegmented.

From the use of the parameters indicated in Section 9.3.1., a room has been undersegmented. Accordingly, the code allows to re-segment any of the obtained rooms with different parameters:

```
answer = input(“Is there a room that needs to be modified? (‘yes’ or ‘no’): “)
```

If the answer is yes and the parameters `voxel_size`, `dist` and `width_door` are modified. Now voxel_size = 0.05 (as the voxel size is smaller, there will be more precision), `dist = 0.01`, and `width_door = 1.3.` Figure 15 shows the result of applying the new parameters.

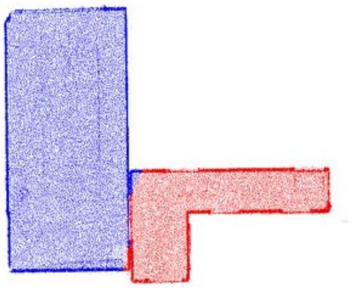


Figure 15. Correctly segmented rooms.

The new results are integrated in the data frame. Because this piece of code is structured as a *while loop*, it will continue asking if modifications are still needed until the answer is `no`.

9.4. Case Study 4

9.4.1. Parameters

- `n_bins: 18`

9.4.2. Implementation

As in the previous case studies, first steps consist of loading the input data, subsampling the point cloud, and removing outdoor points (Figure16).

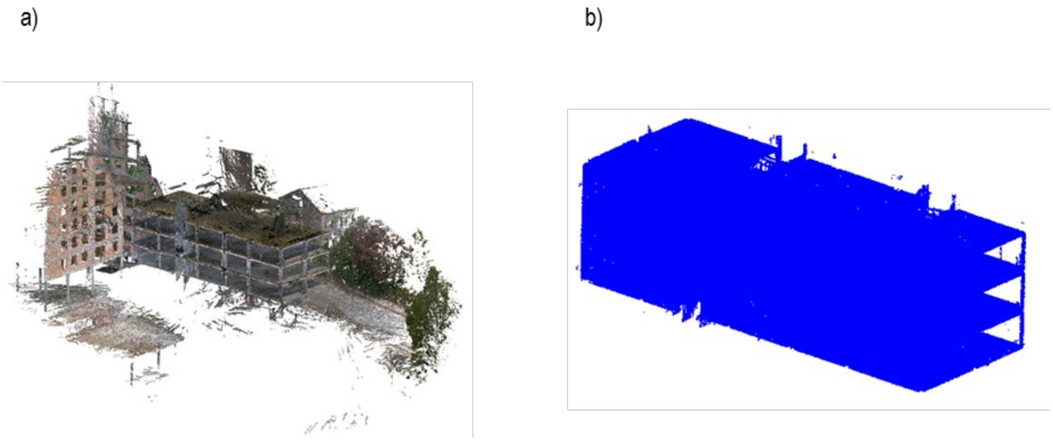
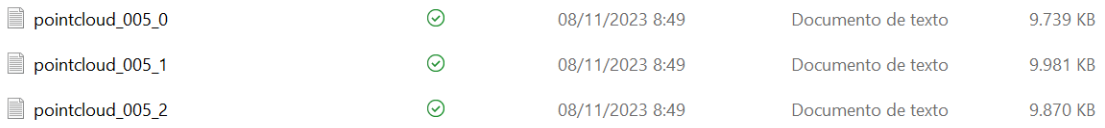


Figure 16. Point cloud a) original, b) manually clean in CloudCompare.

In this case, the point cloud is segmented into three storeys using the method without trajectory (Figure17). The `id_storey` attribute will be added to the three files (in the data frame), but each of them will be assigned a different value, that is, the lower storey will be given a 0, the middle storey a 1 and the upper storey will be a 2. Because this case study is not structured in rooms, further segmentation is not implemented.



pointcloud_005_0	✓	08/11/2023 8:49	Documento de texto	9.739 KB
pointcloud_005_1	✓	08/11/2023 8:49	Documento de texto	9.981 KB
pointcloud_005_2	✓	08/11/2023 8:49	Documento de texto	9.870 KB

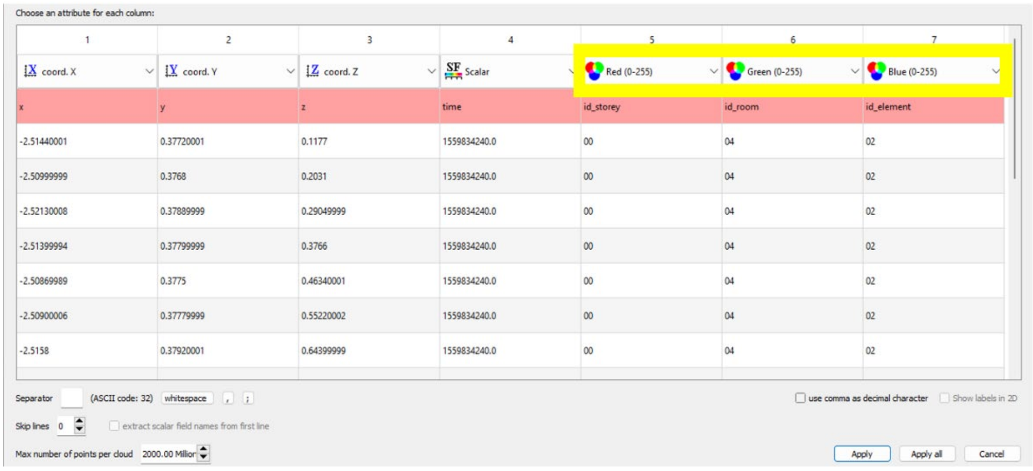
Figure 17. Data frame results in txt format.

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



Choose an attribute for each column:

1	2	3	4	5	6	7
coord. X	coord. Y	coord. Z	Scalar	Red (0-255)	Green (0-255)	Blue (0-255)
x	y	z	time	id_storey	id_room	id_element
-2.51440001	0.37720001	0.1177	1559834240.0	00	04	02
-2.50999999	0.3768	0.2031	1559834240.0	00	04	02
-2.52130008	0.37889999	0.29049999	1559834240.0	00	04	02
-2.51399994	0.37799999	0.3766	1559834240.0	00	04	02
-2.50869989	0.3775	0.46340001	1559834240.0	00	04	02
-2.50900006	0.37779999	0.55220002	1559834240.0	00	04	02
-2.5158	0.37920001	0.64399999	1559834240.0	00	04	02

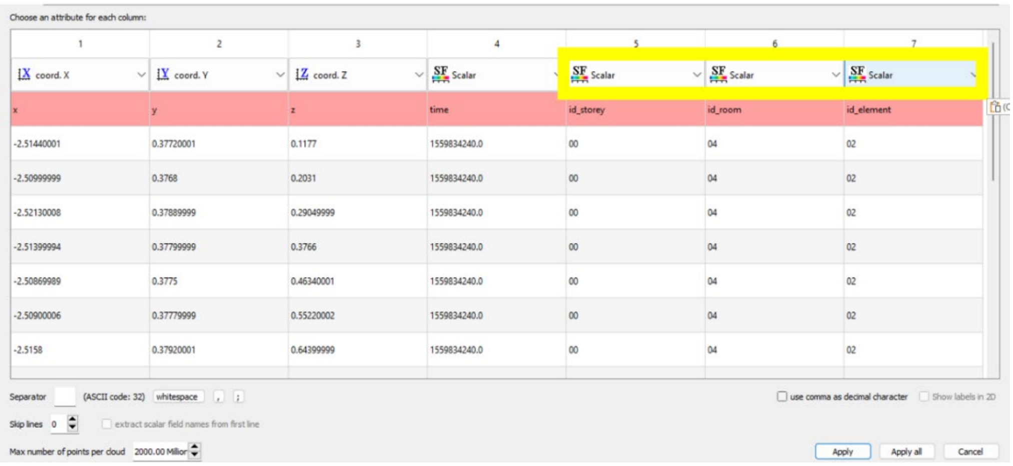
Separator (ASCII code: 32) whitespace , ;

□ use comma as decimal character □ Show labels in 2D

Skip lines 0 □ extract scalar field names from first line

Max number of points per cloud 2000.00 Millior

Figure 18. Red, Green, and Blue default attributes.



Choose an attribute for each column:

1	2	3	4	5	6	7
coord. X	coord. Y	coord. Z	Scalar	Scalar	Scalar	Scalar
x	y	z	time	id_storey	id_room	id_element
-2.51440001	0.37720001	0.1177	1559834240.0	00	04	02
-2.50999999	0.3768	0.2031	1559834240.0	00	04	02
-2.52130008	0.37889999	0.29049999	1559834240.0	00	04	02
-2.51399994	0.37799999	0.3766	1559834240.0	00	04	02
-2.50869989	0.3775	0.46340001	1559834240.0	00	04	02
-2.50900006	0.37779999	0.55220002	1559834240.0	00	04	02
-2.5158	0.37920001	0.64399999	1559834240.0	00	04	02

Separator (ASCII code: 32) whitespace , ;

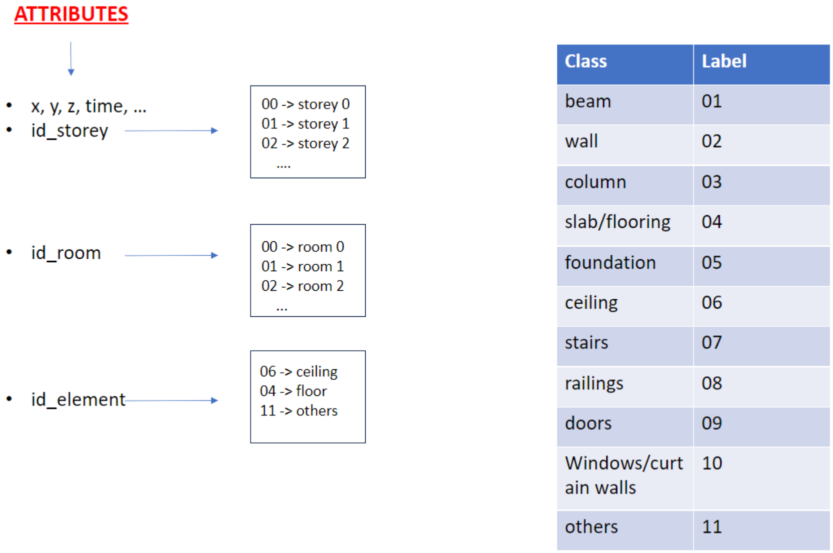
□ use comma as decimal character □ Show labels in 2D

Skip lines 0 □ extract scalar field names from first line

Max number of points per cloud 2000.00 Millior

Figure 19. 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 20. Composition of attributes.

11. Conclusions

This document is intended to provide formal documentation of the code developed in T3.2. The code is tested in four cases, and the purpose of this document is to explain in detail the logic implemented in each of them. All the code corresponding to this 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/B9NtDcMaHnzHoTx.

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.