Otto-von-Guericke University Magdeburg

Faculty of Computer Science

Digital Engineering Project

Automated Transformation of a point cloud to 3D objects

Author:

Sai Dheeraj Reddy Tippani, Sravani Dhara ,Pavan Tummala

April 8, 2021

Advisors:

Dipl.-Phys. M.Sc. Bastian Sander, Dipl.-Ing. Eyk Flechtner
Fraunhofer Institute IFF Magdeburg, Fraunhofer Institute IFF Magdeburg

Dipl.-Ing. Nicole Mencke, Dipl.-Wirtsch.-Ing. Thomas Dengler
Fraunhofer Institute IFF Magdeburg, Fraunhofer Institute IFF Magdeburg

Dipl.-Math. Stefanie Samtleben, Dr.-Ing. Tobias Reggelin
Fraunhofer Institute IFF Magdeburg, Otto von Guericke university Magdeburg

Sravani Dhara, Dheeraj Tippani, Pavan Tummala:
Automated Transformation of point cloud to 3D objects
Digital Engineering Project, Otto-von-Guericke University
Magdeburg, 2021.

Contents

Abstract

With the advent of accessible 3D imaging technology like LIDAR, Stereo imagery etc, the need came up for methods and techniques to convert low level point cloud data into high level CAD data. The current state of the art technologies use Neural networks for both object segmentation and object detection tasks. The fundamental requirement for using such techniques is availability of lot of data to train the model. For scenes like office spaces, the available public data sets are huge in memory size, and demand significant processing power.
Unlike 2D images, the data acquisition of thousands of 3D images is not as easy and straight forward. In this project we have explored the technique of slicing and projecting the point cloud to form a 2D image to use a state-of-the-art 2D image detection technique to extract information about the 'position' and 'identify' the objects in point cloud, with least amount of user intervention.

Acknowledgements

Our team would like to express gratitude to our supervisors Bastian Sander, Nicole Mencke,Thomas Dengler, Eyk Flechtner, Stefanie Samtleben for their extensive support. They have put tremendous effort to guide us in the right direction and gave us insightful information. We thank them for providing us necessary facilities that were required for the completion of project. During the unexpected times our supervisors equipped us with not just the facilities but they also provided huge moral support. A special thanks to Mr. Bastian Sander and Mrs. Nicole Mencke, who have always been a great support with their constant suggestions and provision of resources.

1 Introduction and Motivation

1.1 Introduction

Point clouds are a set of points in space and are acquired from the real-world objects which consists of natural objects like trees, ground and also man made objects like buildings, cars[LI et al. (2018)]. The raw data of point clouds do not have any structure unlike images, which possesses a standard structural unit called pixels. In other terms point clouds do not have any information about the connectivity between its points. Due to this fact, Extracting useful information is a complex problem, even in the case of dense point clouds[SCHNABEL (2010)].
Just like the images point clouds also face problems of varying light conditions and occlusion noise. On top of all these the major challenge when processing point cloud is, its volume. Point cloud data files are large in size, with data of billions of points. If the size is compromised, resulting in a sparse point cloud with lot of missing points which will further make it harder for many algorithms to extract useful information from the data, this directly results in performance issues.
On the other hand, if the size is too large, the processing speed and resources required to process are high. Therefore it is important to find the right balance between the size of point cloud and the appropriate algorithm to perform processing.

1.2 Motivation

'3D Computer Aided Design' models are widely used in numerous engineering and scientific domains. In simple terms these models are helpful in determining how well objects fit together and are mostly used in design

process. Many industrial environments use them for simulation and training [BEY et al. (2012)]. One can say CAD data is something of a high level data with maximum signal and almost no signal. Where as point clouds are raw data forms with a significant proportion of noise. To convert the noisy point cloud data along with noise into noiseless 3D CAD model is the motivation behind this project.
In some applications it is very important to have accurate matching between the actual geometry and 3D CAD. The obtained 3D CAD model would be later used for various applications. Some of such applications that demand precision are:

- Reverse engineering - 3D SCAN (Real world data) to 3D CAD (Idealized data)
- Civil engineering and Architecture
- Maintenance and mapping of buildings, Dams, geological features etc.,
- Computational Fluid Dynamics (simulations)
- Preservation of historically significant buildings, artifacts, etc.,

1.3 Aim of this project

The aim of the project is to develop a pipeline of semi-automated recognition of 3D objects from a point cloud. The initial sketch for the project is as shown in Fig 1.2. The target application is to identify and isolate objects from a point cloud of an office space. One such an office space is shown in Fig 1.1

Figure 1.1: Panoramic image of an office space

The project consists of several sub-tasks. Sub task 1 is to do a literature survey to find the appropriate methods for classification of point cloud data and detection of objects. Sub task 2 is to generate a 3D models from classified objects by the means of template matching. The 'point cloud' objects are replaced by best matched template 'CAD' objects. In the last sub task, the whole process is streamlined into a single program which takes point cloud data as input and export CAD object as output.

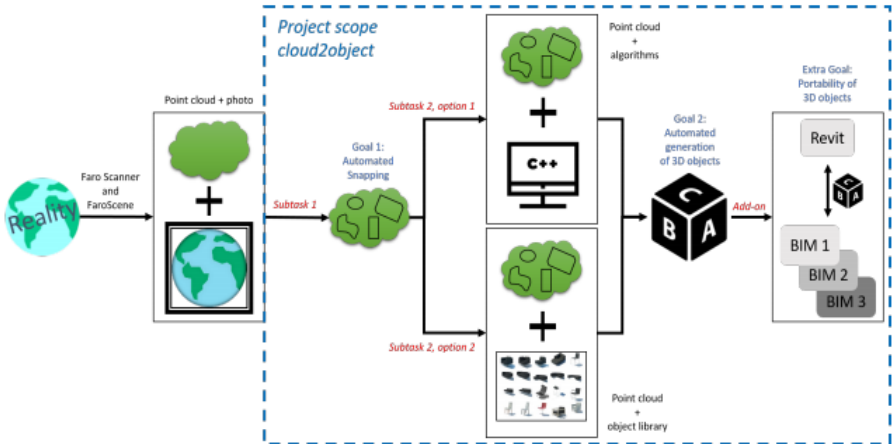


Figure 1.2: Sketch of project cloud2object. The respective subtasks are explained in text.

1.4 Software support

Point cloud processing needs extensive support with respect to the processing and computation. [RUSU and COUSINS (2011)] developed point cloud library(PCL) that can perform state of art algorithms such as filtering, surface reconstruction, registration etc,. This library makes computation of correspondence operation and normal estimation easy to implement.
Open3D is a open-source library developed to aid 3D data manipulation. It supports python language thus making it lot more easier to work with both traditional data manipulation algorithms and neural networks[ZHOU et al. (2018)] based algorithms.

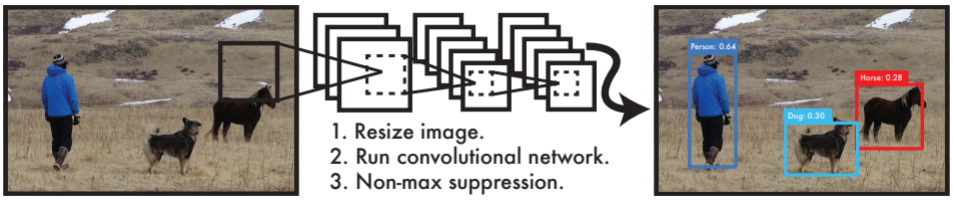


Figure 1.3: YOLO Object detection [REDMON and FARHADI (2018)]

YOLO version3 [RUSU and COUSINS (2011)], a object detection neural network is used. The model can detect objects and export bounding boxes of the detected objects in the image in real time. The algorithm is known for it's speed. With the binary-cross entropy loss function and logistic classifiers, complex objects can also be recognised. Fig 1.3 describes the processing of images by YOLO

Numpy is a python based library, in this project, it is mostly used for simple point-cloud-data manipulation and data handling. [HARRIS et al. (2020)].

'CloudCompare' is a point cloud and mesh processing software. The software is open source. In this project it is being used for converting data from one format to another. Command module mode of 'CloudCompare' software helped in automating the mesh joining operations, near the end of the pipeline.

1.5 Structure of this project report

Chapter 2 will give an overview on the already existing techniques on processing of point clouds and also the algorithms that are developed to perform the slicing operation on the point cloud data. It covers the methods used for obtaining plane from the point cloud data.
Chapter 3 consists of detailed description of the project pipeline and each step in the pipeline.This includes approaches taken to convert point cloud into a 3d CAD object, Various methods performed and the reasons why some of them could not give a standard output are also discussed in Chapter 3. Adding to that, methodology and logical reasoning behind that decisions are explained.

Chapter 4 consists all the approaches explored before ending up with current version of project pipeline. The goal is to justify our currently implemented approach by showcasing the decisions taken during various stages of this project.
Chapter 5 has conclusions from the whole project. The contrast between the initial goal of the project and the output of the project is discussed. From the useful knowledge gained during the project, potential viable directions to take for future developments is discussed in the 'Future Work' section.

2
Related Work

There had been good quality research coming out in the field of point clouds in the recent years . This is because of the latest advancements in the field of point cloud scanners. The laser scanners were now vastly available with a reliable technology and affordable price [RUSU and COUSINS (2011)]. The combination of point cloud data and deep learning approach is growing faster than ever before. Biggest problem with point cloud data is the irregular format, according to [QI et al. (2017)].

Deep learning algorithms use convolutions that indeed require regular input data formats such as image grids or 3D voxels [QI et al. (2017)]. 3D Voxel grids consume system memory that increases exponentially with increase in depth levels of voxels. Smaller voxel size is equivalent to an image with huge number of pixels. More detail will be covered but at a cost of computation and memory. Considering this problem, Pointnet [QI et al. (2017)] were first to develop a network that can take point cloud raw data and process it without using any intermediate conversion. However the spatial data is well used but the RGB data and intensity data were not efficiently used in this method. 'Pointnet' is used in our initial approach and Chapter 4 discusses the pitfalls and challenges faced during the implementation.

Considering the availability of sensors, Terrestrial Laser scanner(TLS) has been widely used in various projects to extract point cloud data. The obtained data is huge and when used for reconstruction it can yield a proper reconstruction of 3D model. [LUO and YAN-MIN (2008)] have proposed a new method which is applied on ancient Chinese buildings. This method involves extracting slices from the point cloud and from the extracted slices, pillar features are found.

This method is interesting because it does not have to hassle with 3D data

processing. Slicing the point cloud and converting a 3D problem into a 2D problem seemed to give promising results with respect to accuracy and at the same time the complexity of the problem is reduced to the great extent. Once the slices are obtained the 2D image processing algorithms would suffice and there are many state of art methods in 2D image processing that produces a satisfying result.

There were various methods to extract features from point cloud, most often used methods are Hough Transform, Random Sample Consensus (RANSAC) and template matching. Out of these methods template matching could extract CAD models from point cloud and is most straight forward method for transforming point clouds into CAD models [LUO and YAN-MIN (2008)] however there are various hyper parameters that are to be tuned in this process. Prior idea on the size of object and other geometrical properties is necessary to create a template for template matching algorithm. This is a tedious process considering the amount of time for creation of template for each object.

According to [LUO and YAN-MIN (2008)] all the points are joined based on Geo-referencing coordinate system with two axis parallel to horizontal plane and one perpendicular to the other two axis as shown in below Fig 2.1. Referencing Fig 2.1, vertical axis of figure is Z-axis, horizontal axis is X-axis. Slices are obtained by cutting along Z-axis with selected intervals. Each slice obtained from cutting is then projected on X-Y plane to obtain the radius of slice. Later based on the value of normal and radius of circle in X-Y plane the cylinders(pillars) are replaced by 3D models. This method performed better than RANSAC and Hough transform in terms of processing time and computational complexity but the algorithm is specific to pillars and cannot be applied on generic basis.

For the detection of planes from point cloud RANSAC showed better performance according to [YANG and FÖRSTNER (2010)] Plane detection is the prerequisite for many computer vision tasks. RANSAC when combined with Minimum description length (MDL) algorithm detection of wrong planes can be avoided. The principle of RANSAC is to find the best plane among 3D point clouds.

The basic idea behind the search algorithm is to randomly choose 3 points then compute plane parameters for the plane constructed from these three points. Now it will detect all points that belong to this new plane. This procedure is repeated and every iteration it compares the new results

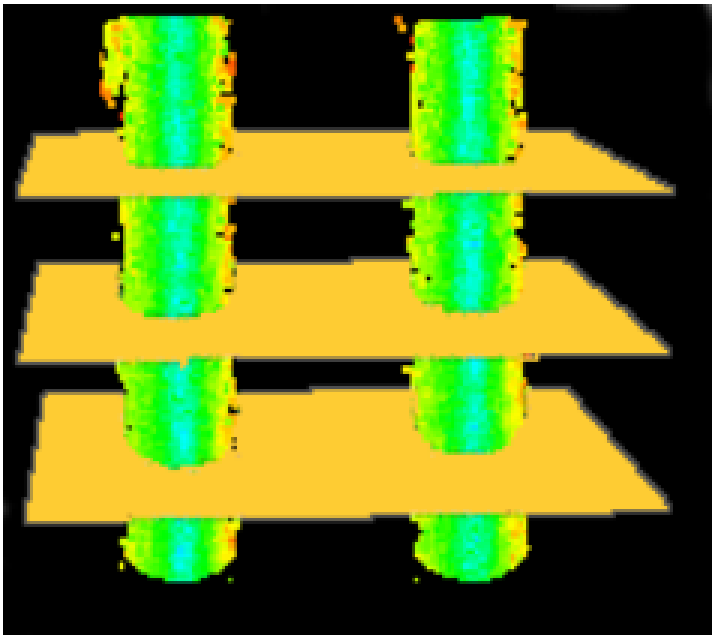


Figure 2.1: Slice of the point cloud(Partial) [LUO and YAN-MIN (2008)]

with the previous one. As per [YANG and FÖRSTNER (2010)] algorithm, the point cloud is separated into small rectangular blocks and in each block RANSAC is applied followed by MDL to ensure there are no wrong planes detected. Later region growing is applied to merge the planes in the limits of threshold provided by user. The pseudo code for this algorithm is shown in Fig 2.2. This method however is not extended to other shapes such as cone or cylinder.

Sometimes the features extracted from the slices play a major role,it is important to define right features from slices obtained by the method of [LUO and YAN-MIN (2008)]. Right features would vary based on the context. In case of building data in [LUO and YAN-MIN (2008)], **radius** feature gives more information than **brightness** feature. Based on the time of photograph brightness will change and it is illumination dependent feature. If such feature takes importance the model would give wrong predictions. Supporting this fact [KYRIAZIS and FUDOS (2008)] developed a method for extracting features from slices. The idea was to find a sub-region in the vast point cloud that can inculcate object characteristics. Initially the point cloud is sliced and in the 2D point cloud slice the curved patches are

Algorithm 2 Proposed algorithm for plane detection

1: partition point cloud into rectangular blocks
2: Assume: a maximum of three planes in each block
3: Initialize: $\Phi_0, \Phi_1, \Phi_2, \Phi_3$
4: **for** each block **do**
5: calculate Φ_0, as in eq. 1.4
6: apply RANSAC 1 to extract a plane
7: calculate Φ_1, as in eq. 1.5
8: remove the points belonging to the first plane
9: apply RANSAC to extract a plane
10: calculate Φ_2, as in eq. 1.6
11: remove the points belonging to the second plane
12: apply RANSAC to extract a plane
13: calculate Φ_3, as in eq. 1.7
14: calculate $i^* = \arg_i \max \frac{exp(-\Phi_i)}{\sum_{i=0}^{3} exp(-\Phi_i)}$, i^* planes detected
15: **end for**

Figure 2.2: Algorithm for plane detection [YANG and FÖRSTNER (2010)]

extracted. These curve patched are then interpolated to form the object boundary using convex hull and voronoi diagram of point cloud[KYRIAZIS and FUDOS (2008)].

3
Methods

This chapter is a detailed walk through along each step that is performed to reach the goal of project. The aim of this Algorithm is to build a function which takes a point cloud file as input, preferably in ".pcd" format and outputs a CAD/Mesh file. Observe Figure 3.1. PCD format is most preferred due to the ease of mathematical operation and most of the 'Python' based libraries have readily developed functions for '.pcd' file objects. The output is a CAD object consisting the input point cloud and certain distinguishable objects as independent CAD objects.

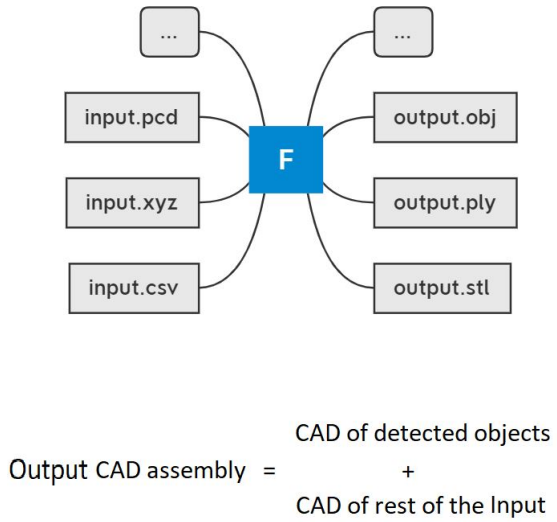


Figure 3.1: A functional representation of the process

.

From the literature survey, [LUO and YAN-MIN (2008)], it is understood that the reliance on 2D images for extracting such information is less com-

plex and not a data-hungry approach.
The 'function' or 'algorithm' extracts useful information, which, in the scope of this project, is the location and identity of individual objects in an office space. The 'useful' pieces of information needed are:

- The identity of the object. Given by object detection algorithms that work on 2d images.
- Location of the center of the object. Extracted by analysing location and time of object detection.
- The size of the objects. An Axis aligned bounding box describing the size.

In the following section, the step by step process involved in obtaining above mentioned information, is given. Further sections will go through each step in more detail.

3.1 Approach

Below are the steps involved in our approach to meet the project goal. After the final step is performed the end result is a 3D CAD file of the input point cloud.

1. Identifying the objects that have to be isolated from the point cloud.
2. Slice the point cloud into "narrow strips", to reduce the influence of neighbouring objects.
3. Convert sliced point cloud into a 2d images.
4. Use state of the art object detection algorithms to detect objects and output a 'boundary box'(BB).
5. Generate real world coordinate(Point cloud co-ordinates) from obtained image detection bounding box.
6. Validate the detections by cross references with other detections.
7. Create a 3d-real bounding box of the confidently detected object.

8. Remove the detected points from the parent/input point cloud.
9. Generate mesh with the updated input point cloud.
10. Assemble the generated mesh with template mesh files of the detected objects.

The Fig 3.2 visualizes some of the steps mentioned above.

Figure 3.2: visualization of the process until extraction of real coordinates of the detected objects

3.1.1 Identifying the objects

The user will choose the objects of interest in the input point cloud. In the case of office space point clouds objects like chairs, desktops, plants, tables etc., and features like doors, windows, lights etc., are selected. These are the objects which the user wants as independent CAD in the final output CAD. During the implementation, we played the role of user and identified some of such objects, which can be found in section 3.1.4.

This specific data will be useful in running slicing operation in a preferred directions/axis. Given a single room point cloud, assuming the walls are parallel to X-Z and Y-Z planes, and the roof and floor are parallel to X-Y

plane, the length and breadth of the room will be represent in the x, y axes and the height is represented in the z axis.
In case of in-applicability of above assumption, the frame of reference must be adjusted accordingly prior to the following steps.

3.1.2 Slicing the point cloud

The operation of slicing a point with various sizes and configurations. Fig 3.4, Fig 3.5, Fig 3.6 and Fig 3.7 show the various configurations we have used in our pipeline. Figure 3.3 below displays how the slices of different sizes cover the length of the point cloud. This variable width slicing will ensure that objects are detected irrespective of their size. Smaller objects get detected in the smaller widths but the larger objects will have only part of it in projection if the width is small. Hence larger width ensures large objects detection.

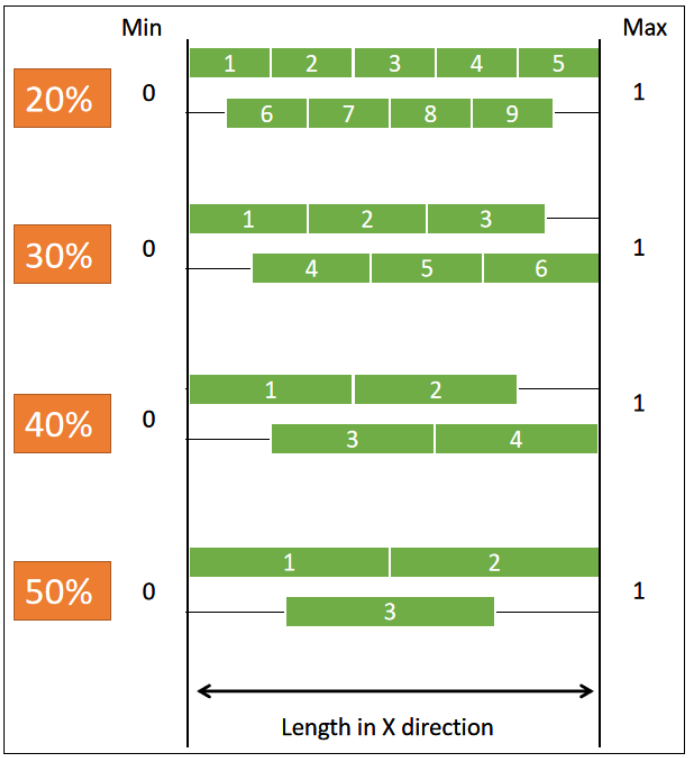


Figure 3.3: Visualising length coverage with various slice sizes

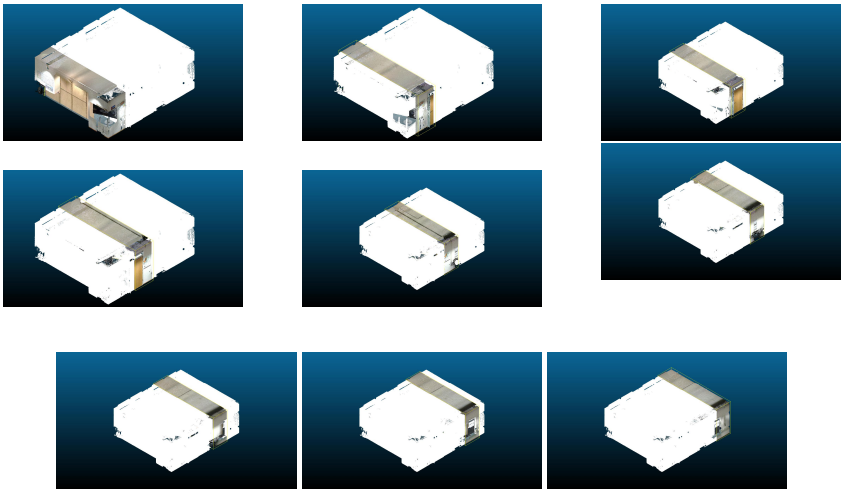


Figure 3.4: Slice size is 20 percent of whole length

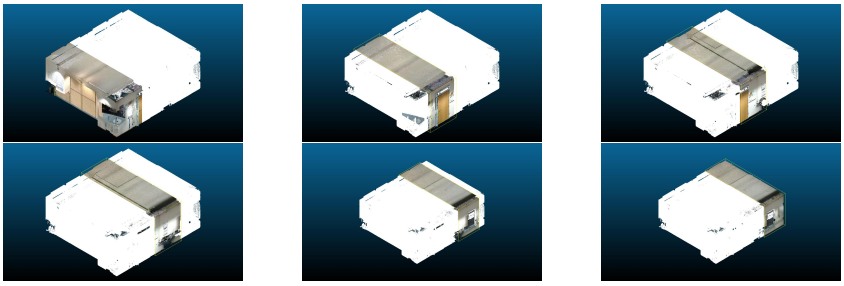


Figure 3.5: Slice size is 30 percent of whole length.

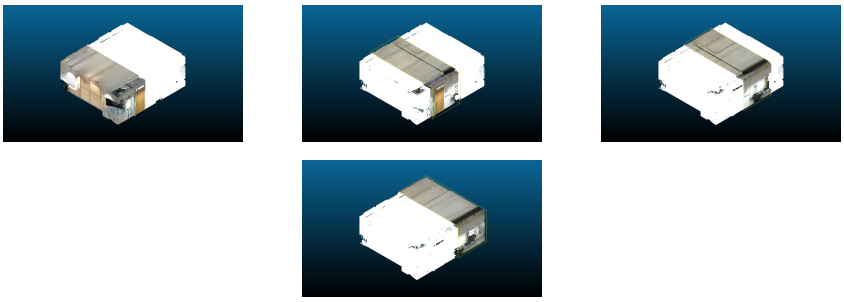


Figure 3.6: Slice size is 40 percent of whole length.

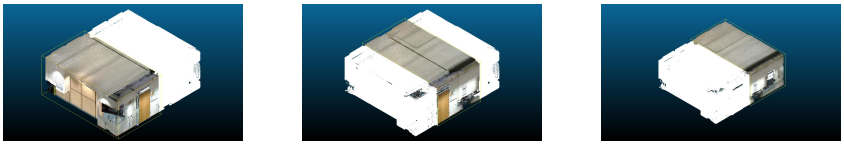


Figure 3.7: Slice size is 50 percent of whole length.

The aim of the slicing strategy here is to ensure that the objects of various sizes and locations will be covered in at-least one of the slice completely. The influence of the background is completely nullified by slicing in a specific size. The points that belong to background wall, are removed in most of the slices. This results in an image with black background, black representing absence of a point perpendicular to the image plane. See 3rd and 4th images in Fig 3.2.

3.1.3 Image generation

Firstly an arbitrary image size is decided by the user. This enables user to decide the best of his interest.
Option1:
determine 'Height',
calculate Aspect Ratio = (Y max-Y min) / (Z max - Z min),
Width = Aspect Ratio x Height
Option2:
Hard coding both Height and Width.
For the sake of explanations say the dimensions of the image are 1000x500 pixels, "Width x Height". The sliced point cloud will be projected onto a 2d image with 1000x500 pixels. The algorithm will iterate through all the points in the sliced point cloud and assign a pixel for each point. The pixel will be assigned the R,G,B values of the respective assigned point. The Fig 3.8 showcases the working of projecting 3d point data onto a plane to form an image(2d).

Figure 3.8: Visualising Projection of a slice into a 2D image

We can imagine the image pixels are bins and the we are trying to map each point into one of the bins based on its (Y,Z) coordinates. When the slices are made in, say, X-axis direction, the resulting image will represent (Y,Z) direction. The Fig 3.9 shows 20% sized slice at various locations.

Figure 3.9: Images generated as the slice moves through the point cloud

When multiple points fall into same bin, the points that are closest to a preferred side of the slice is preferred over other points. This is done to preserve "Opaqueness" in the projection.The Fig 3.10 shows a code snippet of how this process is being implemented.

```
image_array = np.zeros([height, width, 3], dtype=np.uint8)  # empty image sized array
temp = np.full((height, width, 1), xmin)  # used to check if the bin/pixel is already taken
for i in range(sliced_pcd.shape[0]):  # iterating through all points of sliced pcd->(x,y,z,r,g,b)
    w = int(abs(sliced_pcd[:, 1][i] * (width / y_range)))   # mapping point's Y to image's width
    h = int(abs(sliced_pcd[:, 2][i] * (height / z_range)))  # mapping point's Z to image's height
    if temp[h][w][0] < (sliced_pcd[:, 0][i]):
        image_array[h][w][0] = 255 * (sliced_pcd[:, 3][i])  # image's R
        image_array[h][w][1] = 255 * (sliced_pcd[:, 4][i])  # image's G
        image_array[h][w][2] = 255 * (sliced_pcd[:, 5][i])  # image's B
        temp[h][w][0] = (sliced_pcd[:, 0][i])
    else:
        continue
img = Image.fromarray(image_array)
```

Figure 3.10: Image generation by assigning colors of points to the pixels

3.1.4 Object detection

A state-of-the-art object detection algorithm is used to detect the objects that are there in the image that we generated. The object detection algorithm outputs the class/name of the object, a bounding box around the object, and percentage of confidence. The bounding box data will be output in the form of locations of pixels in the diagonal corners of the bounding box, as shown in Fig 3.12. The idea is that the obtained bounding box coordinates contain the information regarding the location of the object in Y-Z space, which we will extract in the later stages.

Figure 3.11: Object detection bounding box visualised

The object detection algorithm we have used is YOLOv3 [RUSU and COUSINS (2011)]. It is a neural network architecture used for real time classification and object detection. We have used a model which is pre-trained on COCO object detection dataset. Of all the classes that the object detector is capable of detecting, we will be needing only a fraction of them. Below are the list of objects that our object detection model will be bound to classify among.
1. Backpack
2. Bookcase
3. Chair
4. Coffee cup
5. Computer keyboard
6. Computer monitor
7. Corded phone
8. Filing cabinet
9. Houseplant
10. Laptop
11. Printer
12. Table
13. Television

The object detection algorithm works as good as the generated image.
The generated image is of better quality(reduced noise) and clarity(no

non-zero points) when
1. Point clouds are dense.
2. Image size is smaller. Example, 100*100 pixels image would have better accuracy than 1000*1000 pixels.
3. The objects are occlusion free
4. Objects are clear enough to detect when viewed from the predefined slice direction.
The mostly black background helps in isolating the object of interest in a given slice.

3.1.5 Extracting coordinates

The object detection model provides us with the coordinates of the bounding box around detected objects. As previously suggested, the bounding box contains the information of the real world (Y,Z) coordinates of the points that belong to detected object.
By using the same mapping formula shown in the code snippet, where we determine exact pixel by (Y,Z) of point, here we extract (Y,Z) coordinates of point , from the exact pixel location of the bounding box corners. The code snippet in the Fig 3.11, we determine the values of (Y corner1, Z corner1) and (Y corner2, Z corner2). At this stage we iterate through all the points in the sliced point cloud and filter out the points which are not inside the range of Y and Z corners.

```
y1_coord = (w1 * y_range) / width  # where y_range is Ymax-Ymin
z1_coord = Z_max - (h1 * z_range) / height  # use Z_max because image_height and Z
# are inversely proportional, Unlike Y and X

y2_coord = (w2 * y_range) / width
z2_coord = Z_max - (h2 * z_range) / height
```

Figure 3.12: Inverse mapping from pixel location to real coordinates

X coordinates of the points that are inside the range will be remembered. The maximum and minimum of all X coordinates we remembered will be our obtained X corner1 and X corner2, respectively. The process is shown in the code snippet Fig 3.13:

```
#for points in POINTCLOUD:
    if (y1_coord) <= y <= (y2_coord):
        if (z1_coord) >= z >= (z2_coord):
            X_values.append(x)
            cropped_point_cloud.append([x,y,z,r,g,b])

x1y1z1_x2y2z2 = [min(X_values), y1_coord, z1_coord,
                 max(X_values), y2_coord, z2_coord]
return x1y1z1_x2y2z2
```

Figure 3.13: Extracting real coordinates

This will look like extrusion of a rectangle (bounding box) to the size of slicing as shown in Fig 3.14 and Fig 3.15.

Figure 3.14: Front view of 3d bounding box

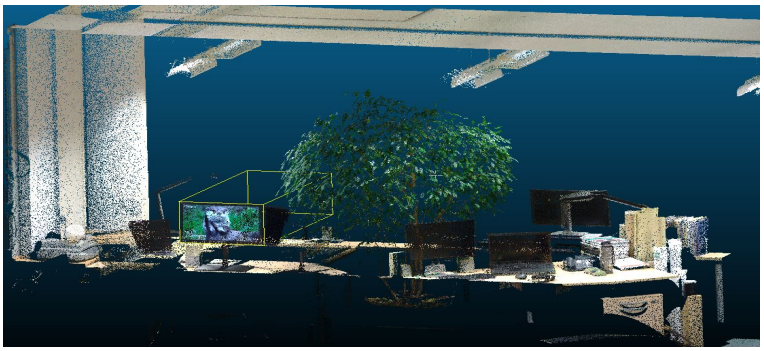
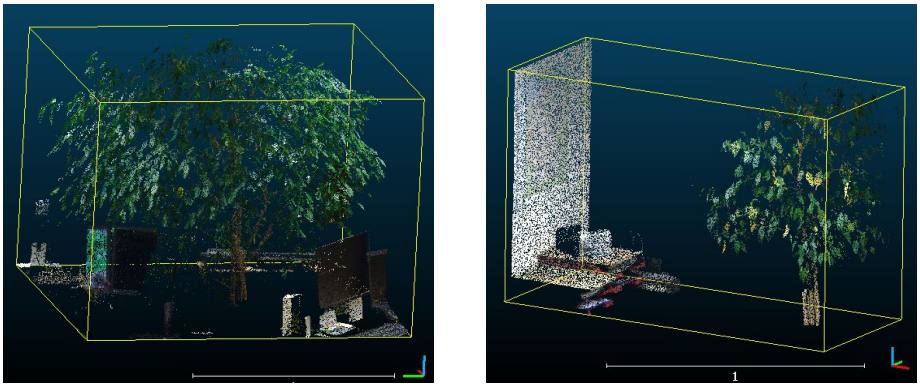


Figure 3.15: Isometric view of 3d bounding box

By the end of this step we will have obtained the locations of diagonal corners of the 3d bounding box. We use this to isolate the points from the parent/input point cloud, so that we can check the correctness of our detection. Images in Fig 3.16 are examples of such detections.



(a) Caption1 (b) Caption 2

Figure 3.16: Detection of potted plant 1 and 2

3.1.6 Removing Duplicate detections

The Object detection algorithm sometimes detects same object in multiple images. Each time the detection and the obtained boundary box location is recorded in a '.csv' file for further scrutiny. Such detections look as shown in the Fig 3.17.

object_name	x1	y1	z1	x2	y2	z2	confidence	image_name
potted plant	-0.022525789	5.439353851	1.977524526	1.404456496	6.040820864	1.014542438	53.55550051	x_sized_20_00
chair	1.874298573	4.929414428	1.046673378	2.86103487	5.465504591	0.660327271	75.142169	x_sized_20_20
laptop	2.132806778	4.988253592	1.063862339	3.588861227	5.452429221	0.677516232	72.50483632	x_sized_20_30
potted plant	2.132770777	2.843892939	1.894218152	3.589307785	4.524077963	0.75824527	79.26952839	x_sized_20_31
potted plant	2.861061573	2.889656734	1.94611539	4.317574024	4.543691018	0.723647111	64.38602805	x_sized_20_40
potted plant	3.589313507	2.876581364	1.941115275	5.045681953	4.609067867	0.701347917	55.71811199	x_sized_20_50
potted plant	-0.022525789	5.432816166	1.971758166	2.132732868	6.021207809	1.020308798	53.8238585	x_sized_30_00
laptop	2.003599405	4.988253592	1.056095886	3.221910477	5.458966906	0.675516139	57.20175505	x_sized_30_10
potted plant	2.246374607	2.896194418	1.923049951	4.317537785	4.556766387	0.821675228	61.93251014	x_sized_30_20
potted plant	-0.022525789	5.439353851	1.971758166	2.783344746	6.034283179	1.026075158	74.53258038	x_sized_40_00

Figure 3.17: Visualising '.csv' file with detection data

The right most column in the image above indicates the name of the image from which the object detection algorithm detected respective objects.

For a human it may be easy to identify the repeated detections by comparing the X,Y,Z values. For example, the object, 'potted plant' is identified in 1,4,5,6,7,9 and 10. But the 'potted plant' in 1,7,10 are not the same plant as rest of the detected potted plants.
To automate this identification process, "Cosine Similarity" is used as a measure to check among detections of same class to uniquely identify the objects. We have considered only the Y1,Z1,Y2,Z2 coordinate values from each detection, since the X1, X2 values depend also on the size of the point cloud slice, which can be misleading if considered for similarity check.
The Fig 3.18 is a matrix representation of identifying the unique objects. The value 1 represents at least 95% of Cosine similarity, while 0 represents anything less than 95% similarity.

```
    object_name        x1        y1  ...        z2  confidence     image_name
0  potted plant -0.022526  5.439354  ...  1.014542   53.555501  x_sized_20_00
3  potted plant  2.132771  2.843893  ...  0.758245   79.269528  x_sized_20_31
4  potted plant  2.861062  2.889657  ...  0.723647   64.386028  x_sized_20_40
5  potted plant  3.589314  2.876581  ...  0.701348   55.718112  x_sized_20_50
6  potted plant -0.022526  5.432816  ...  1.020309   53.823858  x_sized_30_00
8  potted plant  2.246375  2.896194  ...  0.821675   61.932510  x_sized_30_20
9  potted plant -0.022526  5.439354  ...  1.026075   74.532580  x_sized_40_00

[7 rows x 9 columns]


cosine similarity matrix
[[1 0 0 0 1 0 1]
 [0 1 1 1 0 1 0]
 [0 1 1 1 0 1 0]
 [0 1 1 1 0 1 0]
 [1 0 0 0 1 0 1]
 [0 1 1 1 0 1 0]
 [1 0 0 0 1 0 1]]
```

Figure 3.18: Cosine similarity matrix

The similarity check is performed for all the classes/objects detected. The resultant uniquely identified for the above example is in Fig 3.19.

	object name	x1	...	y2	z2
0	potted plant	-0.02252578921616077	...	6.032114972608971	1.0203305238768148
1	potted plant	2.132770776748657	...	4.558618223635002	0.7539779709751062
2	chair	1.8742985725402832	...	5.465504590988159	0.6603272705078127
3	laptop	2.1328067779541016	...	5.455700022175471	0.6765176640598843

Figure 3.19: Unique objects identified

At this point, the 'confidence of detection' and 'image name' are redundant information because all the information needed for the next step is already obtained. The next step is to remove the detected object's points from the parent point cloud.

3.1.7 Removing the detected objects from input point cloud

3D Bounding box of unique objects are now detected. In the input point cloud we remove all the points that fall inside this 3d bounding box. This allows us to replace the object with a pre-existing template CAD model. The updated input point cloud will now contain some empty spaces with no points and some seemingly randomly spilled points. These could be remnants of any poorly cropped out objects.
We perform an outlier removal on the updated input point cloud (parent point cloud) to remove any previously mentioned artifacts. We perform Poisson reconstruction technique on the parent point cloud. This will generate a triangular mesh over all the surfaces using the points as the corners of triangles.

We used Open3D, a python library that supports 3d data manipulation operations. We have used the 'Poisson reconstruction' function to reconstruct mesh for our point cloud. The function takes the point cloud object and 'depth' as inputs and outputs a 'mesh' object. The mesh object can be exported as a '.ply' format file. Depth parameter of the 'Poisson reconstruction' function defines the depth of the octree used for the surface reconstruction. The higher the depth value the more detailed the mesh will be, the computational load will also increase exponentially. The choice of the depth is empirical in our implementation. Depth parameter has to be given by taking the size of the parent/input point cloud, point

density etc. into consideration. In the Fig 3.20 'Poisson reconstruction' for the point cloud is shown.

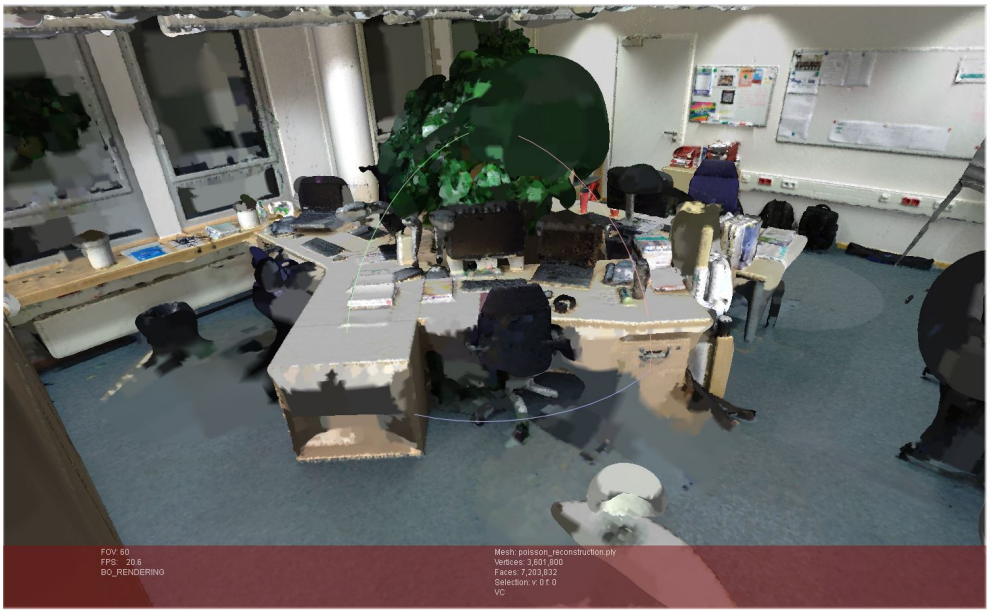


Figure 3.20: Poisson reconstruction

3.1.8 Part Replacement

The final stage of this algorithm is to join a CAD object of the object we have detected and combine it with the CAD of the parent we generated at exactly the same location and orientation, matching the center of 3d bounding box and the center of template CAD object.

The final stage of this algorithm is to replace the empty spaces in the parent/input point cloud, with CAD of the detected objects. The following are the three steps before completely substituting a template cad part with the object in the point cloud.

1. Scaling
2. Finding the center of volume
3. Part replacement

3.1.9 Scaling:

The bounding boxes we have obtained for the uniquely detected objects are Axis aligned bounding boxes. This makes scaling and translation calculations simpler and faster. The template '.ply' objects are unit sized cube objects with variations in color. Each object class is represented by a unique color, this color choice is arbitrary. The same cube object is scaled and used for all unique detections.Scaling can be done either volumetrically or independently to each axis.
Volume based scaling will result in smaller and bigger cubes, which is not very representative of the detected object dimensions and shape.When scaling of the cube is done for each axis independently in such a way that the ratios between length, breadth and height are maintained, this will be a better representation of the bounding box when compared to volume based scaling.

3.1.10 Finding the center of volume:

Axis aligned bounding boxes make it easier to find the center of cuboid with only two extreme corners of the bounding box. The template cube object's center will be transformed to center of the detected object's bounding box.

3.1.11 Part replacement:

Part replacement is done using 'Cloud Compare' command-line mode. A batch file is created which looks as shown in Fig.3.21. The '.bat' file is accessed at the end of the python code pipeline, where the part-replacement is the only remaining step. For this particular step, 'Cloud Compare' software has to be pre-installed in the device in which the project is being run.

```
@echo off
echo "Script running..."
set Import1="...\input_file_path1\mesh_object1.ply"
set Import2="...\input_file_path2\mesh_object2.ply"
set Export="...\output_file_path\combined_mesh_object.ply"
set CloudCompareExe="%ProgramFiles%\CloudCompare\CloudCompare.exe"
%CloudCompareExe% -AUTO_SAVE OFF -O %Import1% -O %Import2% -MERGE_MESHES -M_EXPORT_FMT PLY  -PLY_EXPORT_FMT ASCII -SAVE_MESHES FILE %Export%
echo "Script complete."
```

Figure 3.21: '.bat'(batch file) for using Cloud Compare command line mode

The final Mesh '.ply' will contain the assembled template CAD objects and mesh generated from input point cloud.

4

Experiments and Evaluation

This section will discuss about the other approaches tried. The following approaches were discarded due to reasons like unreliability, processing overload, time consumption etc.

4.1 Segmentation and Correspondence check approach

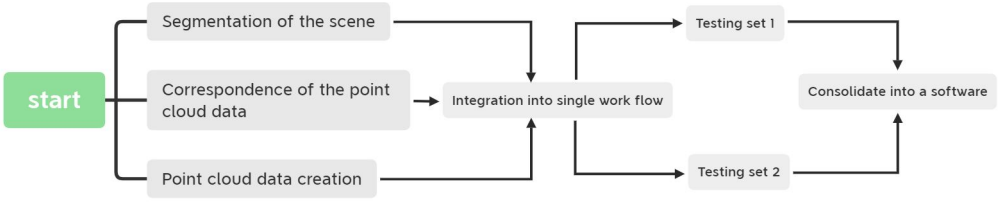


Figure 4.1: Pipeline of 3D data segmentation method

In this initial approach point cloud segmentation is attempted to be achieved using 3D neural network model called Point-Net. The Model is trained on data-sets which are made of pre-annotated point clouds. Post segmentation, the objects of interest in the scene will be isolated to perform a 'Correspondence check' on the object. The following are the methods used for correspondence check. Fig 4.1 shows the considered pipeline. Qi et al. (2017)

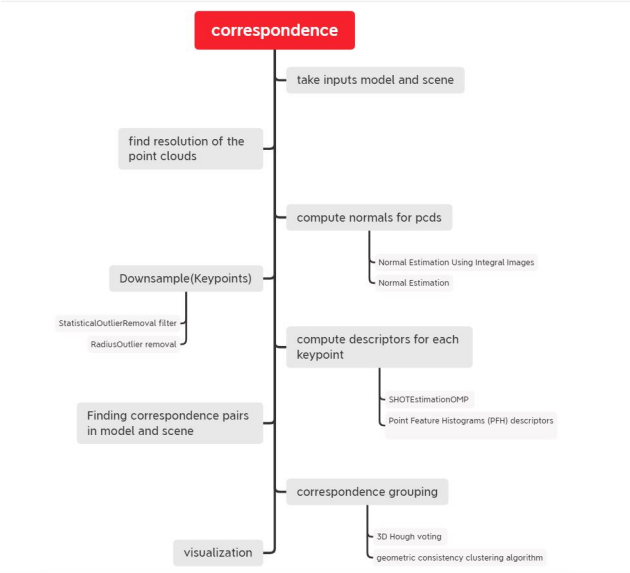


Figure 4.2: Pipeline of 3D data correspondence method

The Goal was to convert the point cloud to a CAD object either from the geometry of points or by object replacing with a template object. The correspondence check is used to suggest a template object which is most suitable the object from scene. Fig 4.2 shows the algorithms used for extracting useful features to find a match/correspondence for the segmented point cloud. RUSU and COUSINS (2011)

The approach is rejected due to the following shortcomings.

- Unavailability of 3D annotated point cloud data sets for office spaces.
- Manual annotation and data collection is time and labour intensive. This makes training a Point Net model very difficult.
- Correspondence check proved to be unreliable and computation intensive.

4.2 Slicing and Projection approach

The current implementation is based on this approach. The idea is to utilise well developed 2d image recognition techniques to extract useful

3D information. This addresses all the previous short comings, there are enormous 2D image data sets readily available. Reliable object recognition techniques output a confidence score for the recognition and pre-trained models are readily available.
The approach transformed multiple times and finally to current implementation. The Fig 4.3 depicts the first rough design of the project pipeline.

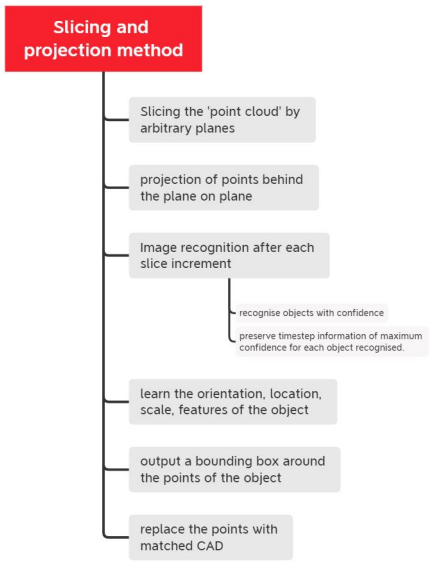


Figure 4.3: Pipeline of slicing and projection method

Initial strategy was to slice the point cloud an arbitrary number of times and project the background onto the slice each time. Repeat such slicing and projection in X, Y and Z directions. Here the major assumption is that the point cloud of office space is axis aligned. If not axis aligned, a transformation/rotation step would have to be applied prior to slicing and projection operation.
The objects in the scene like a chair or a computer, emerge as the slice proceeds in a given direction, for example, in X-axis direction. The objects that emerge first will be recognised in the object detection algorithm first. Given an object, the strategy gives information about where the object starts and ends in the direction X.
Similar projections will be performed in Y-axis and Z-axis directions. Correspondence between detections in all directions must be made automatically to uniquely identify an object.

4.3 Plane detection:

Post detection of objects inside the scene in the point cloud, scene features like walls, doors, windows, roof, floor will have to be modeled so as to fill the space with CAD of objects that have been detected. A plane detection approach is considered to extract useful information about features like walls, roof, floor etc., The algorithms used for plane detection were:

- RANSAC: Random sampling consensus (RANSAC) algorithms is used to extract plane equations for the most densely populated planes in the point cloud.
- Histograms: Histograms for coordinate values of the points in point cloud is made. The idea is that there will be sudden spike in the number of points at walls where the points all share same coordinate value, either x or y or z, depending on whether its a wall or a roof.

This approach is grounded because the cloud density is always the weakest at the walls of the room and strongest in the center of the room. The non homogeneous distribution of points make plane detection too unreliable to extract walls information.

4.4 Up-sampling:

In order to generate high resolution images from the point cloud projections, up-sampling of point clouds is attempted. Unlike random down sampling, random up-sampling doesn't work because up sampling should only be done between surface points in the point cloud.
To address this issue we tried to do reconstruction of a rough surface mesh for the existing point clouds and write a .ply mesh object as output. The .ply object contains surface information in the form of triangles. Generating random points in the plane of each triangle will generate enough points to increase the resolution of the image.

This approach was grounded due to extreme computational load. Instead, the projected image with current non-upscaled point cloud is modified to remove previous artifacts.

4.5 Image color filling:

The image generated from the projection had pixels which were never assigned any color due to low density of the point cloud. To address this problem the pixels which are black, i.e, respective 'RGB' value = (0,0,0), such pixels are assigned the color of arithmetic mean of the nearest 25 neighbours. This neighbour size is arbitrary. The more the neighbours are considered, the higher will be the computation and the image looks blurry and glowing. The Fig 4.4 showcases before and after this filling operation.

Figure 4.4: Images before and after applying color filter on zero elements

This approach is also computationally intensive but the result is worth it due to our dependency on object recognition in the steps that follow image generation.

4.6 Sized Slicing:

The idea of sized slicing is to remove the extra step of projecting slices in multiple directions, by extracting maximum amount of information through one direction of slicing alone. The catch is that the size of the slice is discretely varying. The rate of variation is arbitrarily chosen to demonstrate the applicability of the approach. The approach is address in more detail in Chapter 3.

5 Conclusions and Future Work

5.1 Conclusions

Various methods and also multiple combinations of algorithms are tried among which the best one proved to be the conversion of point cloud into slices then applying object detection algorithm(YOLO in our case) later replace the object with its CAD model. This has simplified the complex 3D problem into a 2D problem along with that it saves lot of memory during processing. Since the intermediate point cloud results, computation with Numpy arrays, point co-ordinates are not saved, so there is no requirement to allocate huge memory to save the intermediate results.

Two methods have been proposed. Out of which first method needs prior knowledge regarding the characteristics of point cloud like size and density to compute the hyper parameters that should be given as input along with point cloud. In second method which is the main method it is ensured that there is minimal manual work and have the complete pipeline setup without any manual intervention. Complete pipeline has been represented in the flowchart Fig 5.1

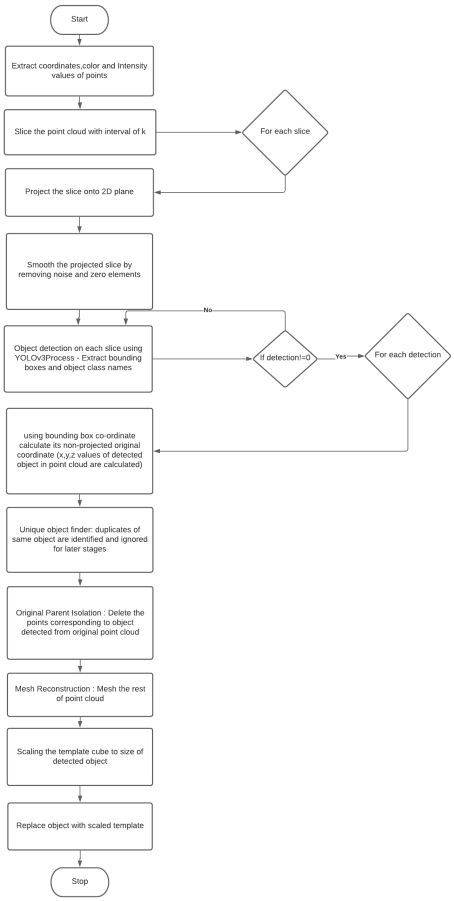


Figure 5.1: Complete flowchart of method 2

Only the point cloud should be given as input and the rest is taken care by the program.

5.2 Future Work

5.2.1 Object detection

The state-of-the-art YOLOV3 algorithm is pre-trained with coco-image dataset. By using a pre-trained model the computational load is drastically reduced. But the model trained by this data, might not represent our use case accurately. The image generated from point clouds tend to possess certain level of artifacts. Removing such artifacts is computationally

costly. This is discussed in Chapter 4.5. Potential alternatives for the sake of better object detection are:

- Custom train YOLOV3 with images that are generated from point clouds.
- Exploring alternatives to YOLOV3
- Removal of image artifacts by using efficient methods.
- Use 'object segmentation' algorithms instead of 'object detection' algorithms
- Up-sampling point cloud for generation of denser images.

5.2.2 Slices

The slice sizes implemented in the current project are empirically chosen. There reasoning is to roughly perform enough slice operations to demonstrate the idea of information extraction from images. The following are the ideas that will help in empirically choosing the slice configurations for future work:

- Increased number of slices will increase confidence of detected object.
- Reduction in slice sizes will increase accuracy of depth prediction of detected object.
- Smaller sizes and more slices will increase computation load.

5.2.3 Programming language

The current implementation is Python based and is heavily dependent on the point cloud and mesh data manipulation libraries. Point cloud and mesh data manipulation demands high computation and hence choosing a programming language which will increase the speed of computation drastically. For example C++.

5.2.4 General comments

Just like every coin has two faces this algorithm also has points of improvement. The first and foremost would be regarding the performance of object detection algorithm. As discussed in section 3.1.4 the classes of YOLO at present are very minimal and if provided with extensive training with other classes the performance can increase to great extent.
Second point of improvement would be at the slices. Since the point clouds are un-ordered data, they have lot of Noise. If the sliced point cloud images are smoothed then there is better chance of optimal object detection. It is to be noted that over smoothing may blur the image and the object boundaries may be lost in this case.

5.3 Workload

For the sake of better clarity on distribution of work among the team, the following image 5.2 describes the roles that each team mate played.

	Dheeraj	Sravani	Pavan
Literature review	25	50	25
coding	60	35	5
Research-brain storm	40	40	20
Report	40	60	0

Figure 5.2: Contribution chart

A

Abbreviations and Notations

Dataset and clustering acronyms

Acronym	Meaning
PCL	Point Cloud Library
RANSAC	Random sample consensus
TLS	Terrestrial Laser scanner
YOLO	You only look once
CAD	Computer-aided design
3D	Three dimensional
Xmin	Minimum of X value
Xmax	Maximum of X value
Ymin	Minimum of Y value
Ymax	Maximum of Y value
Zmin	Minimum of Z value
Zmax	Maximum of Z value
h	Height
w	Width

B

List of Figures

C

List of Algorithms

1.RANSAC - Random sample consensus
2.MDL - Minimum Description length
3.PointNet
4.YOLO - You only look once

D
Bibliography

[LI et al.] **PointCNN: Convolution On $\mathcal{X}$-Transformed Points**.

[SCHNABEL 2010] R. Schnabel. **Efficient Point-Cloud Processing with Primitive Shapes**. Dissertation, Universität Bonn, 2010.

[BEY et al. 2012] A. Bey, R. Chaine, R. Marc, G. Thibault and S. Akkouche. **Reconstruction of consistent 3D CAD models from point cloud data using a priori CAD models**. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXVIII-5/W12, 2012.

[RUSU and COUSINS 2011] R. B. Rusu and S. Cousins. **3D is here: Point Cloud Library (PCL)**. In: 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 1–4.

[ZHOU et al. 2018] Q.-Y. Zhou, J. Park and V. Koltun. **Open3D: A Modern Library for 3D Data Processing**. arXiv:1801.09847, 2018.

[REDMON and FARHADI] **YOLOv3: An Incremental Improvement**.

[HARRIS et al. 2020] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke and T. E. Oliphant. **Array programming with NumPy**. Nature, Vol. 585(7825):357–362, 2020.

[QI et al.] **PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation**.

[LUO and YAN-MIN 2008] D. Luo and W. Yan-min. **Rapid extracting pillars by slicing point clouds**. International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. 37, 2008.

[YANG and FÖRSTNER] **Plane Detection in Point Cloud Data**.

[KYRIAZIS and FUDOS 2008] I. Kyriazis and I. Fudos. **Identifying features by slicing point clouds**. 2008.

Declaration of Academic Integrity

We hereby declare that we have written the present work ourselves and did not use any sources or tools other than the ones indicated.

Datum: .....
(Signature)

Datum: .....
(Signature)

Datum: .....
(Signature)