

**School of Computing**

FACULTY OF ENGINEERING



**UNIVERSITY OF LEEDS**

---

**Deep Learning Approach To RGB-D Object Detection**

**Abdullateef Shittu**

**Submitted in accordance with the requirements for the degree of  
Msc in Advanced Computer Science: Artificial Intelligence**

**Session 2019/2020**

The candidate confirms that the following have been submitted:

<b>Items</b>	<b>Format</b>	<b>Recipient(s) and Date</b>
<i>Deliverables 1</i>	<i>Report</i>	<i>SSO (30/08/20)</i>
<i>Deliverable 2</i>	<i>Software codes or URL on github</i>	<i>Supervisor, assessor 30/08/20)</i>
<i>Deliverable 3</i>	<i>Data on google drive</i>	<i>Supervisor, assesor (30/08/20)</i>

Type of Project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student): ABDULLATEEF SHITTU

## **Summary**

Deep learning approaches to computer vision problems have led to many exciting breakthroughs in the field. This project examines the advancement and the different methods for object detection in 3D.

The project implemented a state of the art, end to end object detection network called VoteNet using Pytorch. Using SUN RGB-B as the indoor dataset, the network was able to detect unordered point clouds and predict bounding boxes.

## **Acknowledgements**

Firstly, I would like to thank my supervisor, Dr. Mehmet Dogar, who helped me tremendously in getting over problems encountered in the project.

Secondly, I would like to thank the University of Leeds for being understanding during the difficult period of the pandemic.

And lastly, I would like to thank my family for supporting me throughout this process, without them I could not have gotten to this point.

## Table of Contents

<b>Summary.....</b>	<b>iii</b>
<b>Acknowledgements.....</b>	<b>iv</b>
<b>Table of Contents .....</b>	<b>v</b>
<b>List of Figures .....</b>	<b>vii</b>
<b>List of Tables .....</b>	<b>viii</b>
<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Project Aim.....	1
1.2 Objectives .....	1
1.3 Deliverables .....	2
<b>Chapter 2 Background Research.....</b>	<b>3</b>
2.1 Literature Survey.....	3
2.1.1 Recent developments in 3D object detection using deep neural networks.....	3
2.1.2 PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation .....	5
2.2 Methods and Techniques .....	7
2.2.1 ROS + PCL + OpenCV.....	7
2.2.2 Image based processing method .....	10
2.2.3 Point cloud based processing method.....	11
2.2.4 Multimodal Fusion based .....	13
2.3 Choice of Methods .....	14
<b>Chapter 3 Background Research.....</b>	<b>16</b>
3.1 Software requirements .....	16
3.2 System Design .....	16
3.2.1 Multimodal Fusion based .....	17
3.2.2 Technologies used .....	19
<b>Chapter 4 Software Implementation .....</b>	<b>21</b>
4.1 ROS Implementation of object detection using linemod .....	21
4.2 VoteNet Implementation.....	22
4.2.1 SUN RGB-D dataset preparation .....	22
4.2.2 Training VoteNet based model.....	23

<b>Chapter 5 Software Testing and Evaluation .....</b>	<b>32</b>
5.1 Testing .....	32
5.2 Evaluation .....	32
5.3 Results .....	33
<b>Chapter 6 Conclusion .....</b>	<b>35</b>
6.1 Conclusion .....	35
6.2 Further Research .....	35
<b>List of References .....</b>	<b>216</b>
<b>Appendix A External Materials.....</b>	<b>422</b>
A.1 External Materials .....	42
A.2 Code For Project on Google Colab .....	42
<b>Appendix B Ethical Issues Addressed .....</b>	<b>43</b>

## List of Figures

<b>Figure 1.</b> Example of 3D Bounding Boxes [11].....	4
<b>Figure 2.</b> PointNet Application [2].....	6
<b>Figure 3.</b> PointNet Architecture [2].....	7
<b>Figure 4.</b> A generic Shape Detector ROS graph [19].....	8
<b>Figure 5.</b> RANSAC Algorithm [38].....	10
<b>Figure 6.</b> Generic CNN [42].....	11
<b>Figure 7.</b> Illustration of 3D Voxelization [10].....	12
<b>Figure 8.</b> Illustration of 3D Convolution [10].....	13
<b>Figure 9.</b> Multimodal Fusion based Network Architecture [23].....	14
<b>Figure 10.</b> Illustration of VoteNet Architecture [1].....	17
<b>Figure 11.</b> Code Snippet of Voting_module.py [1].....	18
<b>Figure 12.</b> Images coupled with their annotations of SUN RGB-D Dataset [31].....	22
<b>Figure 13.</b> Code snippet of Train.py [1].....	24
<b>Figure 14.</b> Code snippet of test_run.py [1].....	25
<b>Figure 15.</b> Code snippet of ap_helper.py [1][44].....	26
<b>Figure 16.</b> Point clouds of 000001pc.ply input file.....	27
<b>Figure 17.</b> Results with predicted bounding boxes of 000001pc.ply.....	28
<b>Figure 18.</b> 000001_pred_map_cls.....	28
<b>Figure 19.</b> xtionPRO live depth images on rqt.....	29
<b>Figure 20.</b> RViz.....	30
<b>Figure 21.</b> Point clouds on RViz.....	30
<b>Figure 22.</b> IOU equation [36].....	32

## List of Tables

<b>Table 1.</b> Technologies used for data processing.....	19
<b>Table 2.</b> Technologies used for evaluating model.....	19
<b>Table 3.</b> Technologies used for visualization.....	20
<b>Table 4.</b> Results for IOU threshold 0.25.....	33
<b>Table 5.</b> Results for IOU threshold 0.5.....	34



# **Chapter 1**

## **Introduction**

### **1.1. Project Aim**

The aim of this project is to investigate object detection in 3D environments using point clouds and implement a deep learning approach to object detection with emphasis on indoor environment. Architectures presently available to perform object detection using point clouds still rely on methodology used in 2D object detection and their neural network architecture [1]. This project will make use of VoteNet which gives great results in comparison to other methods, it doesn't rely on RGB images [1].

Usually, the input data formats for object detection architectures are 3D voxels or image grids for a plethora of kernel optimizations but doing this with point clouds results in a large collection of data that becomes computationally expensive to manipulate [2]. The method used in this project uses a different input representation for dealing with point clouds used in PointNet deep neural network [2].

There are many applications of object detection using depth information. In robotics, the depth information is used by the robot to manipulate objects and have an enhanced interaction with its environment [3], in healthcare, depth information is used to provide an interactive care system for patients suffering from dementia [4], in autonomous driving to inform the car of obstacles and increase safety and in the gaming industry, to create augmented realities for a thrilling user experience. This creates incentives to interpret point data accurately and efficiently in real time.

With the rise of RGB-D sensors such as Kinect and Asus XtionPRO Live, that gives a real time depth information, developers have been able to create computer vision algorithms to retrieve information from the depth information and use that information to detect objects.

### **1.2. Objectives**

- Find appropriate Dataset to be used to train the neural network model
- Train and test neural network with the dataset
- Evaluate model
- Detect objects using point clouds and enclose detected objects with bounded boxes
- Visualize objects in Meshlab

### **1.3. Deliverables**

1. A github repository that hosts the source code and readme file
2. A google drive link that contains Data files
3. MSc dissertation report

## Chapter 2

### Background Research

#### 2.1 Literature Survey

This chapter provides a survey of research tackling the same problem of 3D object detection. It also showcases background research and knowledge acquired before I attempted to tackle the problem.

My survey looked at 2 specific areas:-

- Recent developments in 3D object detection using Deep learning.
- Solution for handling raw unordered point clouds, PointNet.

##### 2.1.1 Recent developments in 3D object detection using deep neural networks

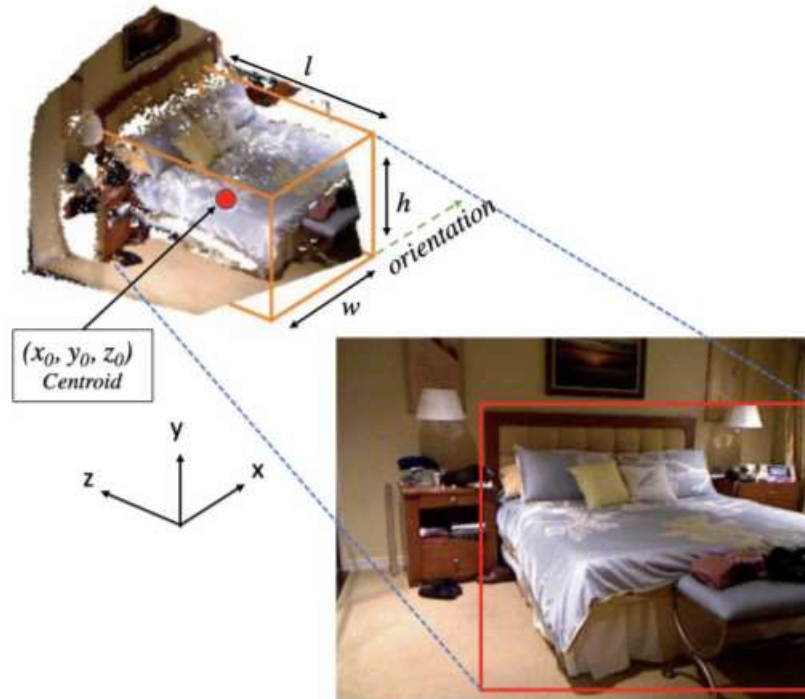
3D object detection is less researched than 2D object detection which has seen more advancements because of the staggering amount of research done in the 2D space [11]. According to M M Rahman et al paper "*Recent Advances in 3D Object Detection in the Era of Deep Neural Networks: A Survey*," [11], There exist several challenges in 3d object detection pertaining to occlusion, fluctuations in scale, variations in view-points, limited datasets on the 3D environment of interest, and data processing. With data processing, they stated that another dimension is needed to handle the depth information which leads to a higher computational cost [11]. The research also states that because of the rapid growth of RGBD cameras and sensors, it has been difficult to keep up with the trends and advancements in the space of 3D object detection [11].

The research carried out comparative analysis and outlined the various state of the art methods and their limitations used to carry out this technology, by grouping the methods into three main categories based on their input modality [11], the categories are image based processing methods, point cloud based processing methods and multimodal fusion based methods [11]. More on the methods in the section 2.2.

The research [11] went into detail about 3d bounding box encoding methods like axis aligned 3D center offset method, 8-corners method and 4-corner 2-height method.

#### 3D Bounding Boxes

3D Bounding boxes are rectangular cuboids in shape placed around target objects or detected objects in 3D co-ordinates.



**Figure 23.** Example of 3D Bounding Boxes [11]

#### **Axis aligned 3D centre offset method**

In recognition tasks involving this method, 3D space is encoded using a directional Truncated Signed Distance Function (TSDF) [39]. The space is structured into a 3D voxel grid with equally spaced cells. Voxels have a value corresponding to the shortest distance between the voxels centre and the depth map [39]. In this method, size and center dimensions are represented in 3D (i.e.  $c_i, c_j, c_k$ ), this along with ground truth annotations and anchor orientation, can be used to represent 3D boxes. The offset method calculates the difference between a positive anchor and its ground truth and regresses to axis aligned 3D boxes [39].

#### **8-corners method**

Unlike axis aligned offset method, multi task loss is used to predict bounding boxes and an IOU overlap of the boxes from a bird's eye view. An IOU threshold is used to classify or propose point clouds of interests [40]. According to Chen et al. this method provides better results than the axis aligned method.

#### **4-corner 2-height method**

Ku et al. noted a drawback with the 8-corners method, stating that the method does not consider the physical constraints of 3D bounding boxes since top corners of the cuboid are aligned with the bottom corners of the cuboid. This method takes a different approach, it encodes the boxes by representing the offset of the cuboid corners from the ground plane using four corners and 2 heights dimensions [41].

Current tools used to collect input modalities and their limitations were discussed, tools like **Monocular Cameras** which capture images, these images infer an RGB 2D image of the 3D target environment. The image contains information regarding texture, color and other appearance features. A major limitation of these cameras are that they are affected by lighting and weather variables and they don't perform well with objects that have little differentiation in color and texture [11], this has led to the rise in thermal cameras which are less affected by lighting and weather variables. **LiDARs** which stands for light detection and ranging deal with varying light intensity and weather conditions better than monocular cameras, they predict distance between objects and the sensor accurately [11], an advancement here is the introduction of flash LiDARs. **RGB-D cameras**, these cameras are different from monocular cameras in that these cameras capture RGB images and per pixel depth simultaneously [11]. All though a downside to this camera is that it can only measure distance of 8meters while LiDARs have a range of 200meters. **Radars** and **Ultrasonics** are also sensors used to collect depth information however they are more challenging and poor in performance when compared to other sensors [11].

Rahman et al stressed the importance of data in 3D object detection. They investigated popular datasets used for training and evaluating current methods and performed comparative analysis on the datasets. More on datasets in chapter 3. The research [11] evaluated the best methods for 3D object detection on KITTI, SUN RGB-D, NYUv2 dataset. These datasets are benchmark datasets. The research paper used intersection over union as their evaluation metric, this is measured as the intersection of ground truth bounded box and the predicted bounded box divided by their union [11].

The research made a prediction that a different trend would emerge due to Qi, et al's paper, "*Pointnet: Deep learning on point sets for 3D classification and segmentation*," [2] which introduced a new way for handling unordered point clouds, the PointNet architecture, a technique which was part of the solution in other to achieve this project's aim [11]. This new technique makes it easier and very efficient in terms of computational cost and increases performance. The old technique was to structure the unordered point clouds in to 3D voxels or 2D projection, very challenging and not cost efficient [11]. So their prediction seems like a safe bet.

This paper concludes by stating that 3D object detection needs more research so as to reach the level of accuracy and reliability as 2D object detection. A lot of opportunities for future work, especially in creating systems that can perform both in outdoor and indoor scenes [11].

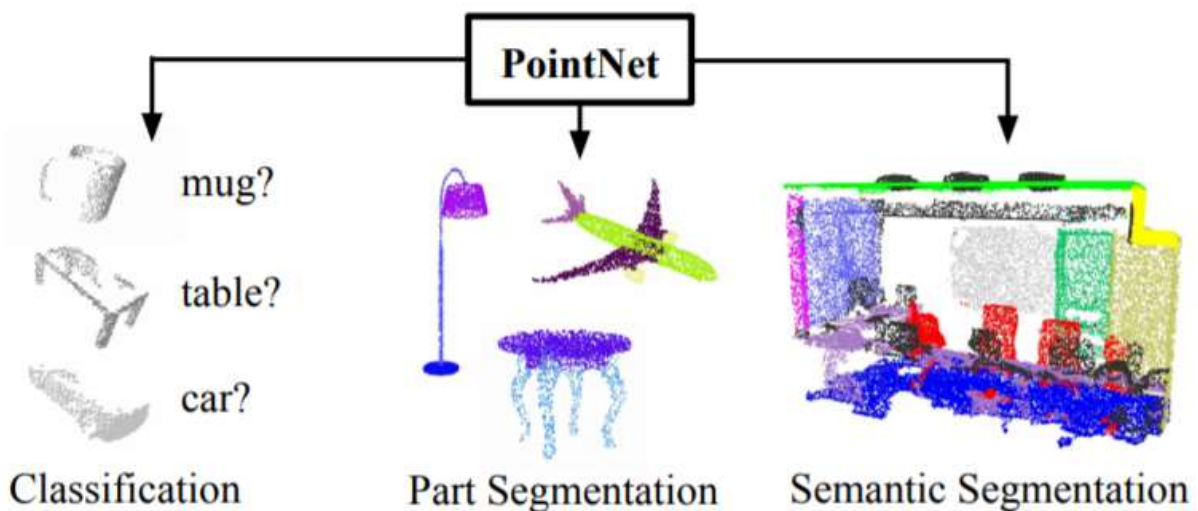
### **2.1.2 PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation**

Point clouds are a collection of data points in space [12] and can be represented in cartesian coordinates (X,Y,Z). points clouds can either be ordered or unordered point clouds. Point

clouds that are said to be ordered, have points that have relationships with each other or for example have vectors connecting each other [13] while unordered point cloud is the opposite, points have no relationships, at least to the observer [13].

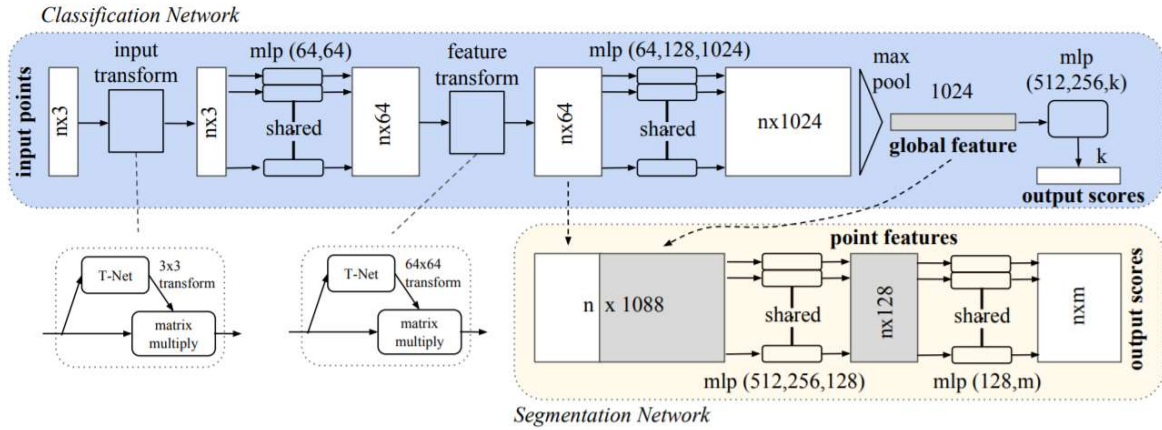
As mentioned in the earlier chapter the technique that Qi et al [2] introduced was the PointNet architecture. In their research paper they state that PointNet can handle unordered point clouds very well, and not by discretizing point clouds and generating 3D voxels as is the traditional way. A downside to the traditional method mentioned in Rahman et al [11] is that it can cause information loss, so it's not all that accurate. To tackle the problem of unordered point cloud data Qi et al [2] developed an architecture to handle the problems posed by data sets [14] with respect to permutation invariance [2].

Permutation invariance refers to when the network identifies the point cloud data of an object as the same whether or not the point clouds are rotated or have gone through rigid transformations [14]. The PointNet network is able to detect relationships between points of the same data set. PointNet network can perform object classification which outputs class labels, part segmentation which outputs part labels per input point and semantic segmentation which outputs segment per input point [2] with efficient performance and produces state of the art results.



**Figure 24.** PointNet Application [2]

Qi et al's network is intuitive, it comprises of a shared multi-layer perceptron to encode points twice from their X,Y,Z coordinates to 64 dimensions and from 64 dimensions to 1024 dimensions [14], then it uses a single symmetric function and max pooling, features are extracted from the points and encoded and passed through final fully connected layers make the classification either shape classification in the case of object classification or shape segmentation in the case of segmentation tasks [2]. [diagram]



**Figure 25.** PointNet Architecture [2].

Their network was evaluated on the ModelNet40 [15] CAD benchmark dataset. This dataset has 40 object classes with 9843 for training and 2468 for testing [2]. Their network in comparison to previous methods scored high [2]. ShapeNet [16] which has 16 shape classes and 16881 shapes was used to evaluate their object part segmentation model and for their semantic segmentation their network was evaluated on the Stanford 3D dataset. This dataset has 3D scans of 13 classes [2]. All evaluations had great results on all datasets.

Their network was tested for robustness and it was confirmed that the network is robust to different input irregularities [2]. With all these advantages over current methods, it's no surprise it was used as a backbone for other 3D architectures like VoteNet, which we'll discuss in detail further as it was used as the solution for this project.

## 2.2 Methods and Techniques

There are different approaches or solution principles to tackle the problem of 3D object detection. The Deep learning approach, which is the current trend and as mentioned earlier in section 2.1.1, the methods using a deep learning approach can be grouped into 3 categories [11] using their input modalities. There is also the approach of using Robot Operating System (ROS) together with point cloud library (PCL) and OpenCV technologies. This section will discuss these methods starting with the latter approach and then the 3 ways deep learning approach can be used for 3D object detection.

### 2.2.1 ROS + PCL + OpenCV

#### Overview

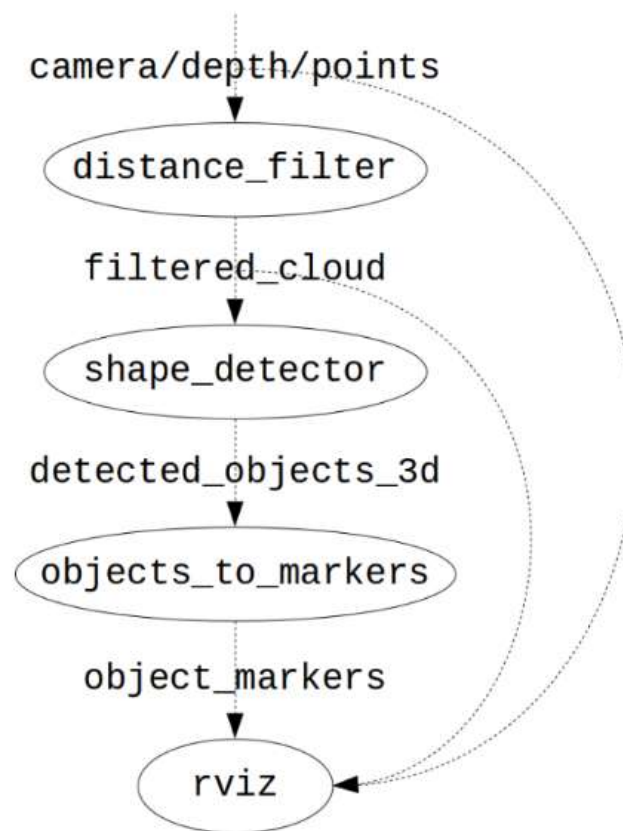
**Point Cloud Library** is an open source resource for the computer vision community [17]. It has a plethora of tools and state of the art algorithms for 3D object detection tasks, from processing 3D data, segmentation and detection [17].

**OpenCV** is another open resource library that contains a collection of tools and algorithm for real time image processing [18].

**ROS** is an open source framework that contains tools for programming robot software [19]. It is a system of nodes called ROS nodes that run at the same time together and can pass information between one another using ROS topics, ROS services and ROS actions [19].

- ROS topics allows Other nodes to subscribe to a node for information or publish information to other nodes [19].
- ROS services allows nodes to execute functions in other nodes
- ROS actions are used for temporal goal-oriented operations

All correlations and interactions between nodes are shown on a ROS graph.



**Figure 26.** A generic Shape Detector ROS graph [19]

ROS has packages that have related tools for a particular tasks. The packages are created in a workspace containing directories and code related to ROS. For example, the tf ROS package which is used to perform transformations with points, can track multiple coordinate frames by using a tree structure to maintain relationships between them [20].

ROS systems are launched using ROS commands, rosrn and roslaunch. Roslaunch launches all related nodes of a system at once while rosrn launches nodes separately [19].



All ROS systems contain ROS masters that nodes report too. Whenever a node subscribes or publishes to a topic, they report that action to their ROS masters [19].

### **Shape Detection using RANSAC algorithm**

Detecting shapes is a basic foundation to 3D object detection, RANSAC and Hough transform are the most popular algorithms for shape detection.

RANSAC is used to detect shapes in unordered point clouds using random sampling, it reduces point clouds into two sets of inherent shapes and left over points [38], the shapes act as proxies correlating to a set of points. The algorithm is robust because it is able to extract shapes and planes from given data regardless of noise [38]. A drawback with this algorithm is that it had to be coupled with other optimizers to save computational cost [38].

Shapes are extracted in these steps [38]:-

- Minimal set are chosen at random from point cloud data (Note: Minimal set refers to the smallest number of points needed to identify with a primitive shape) [38]
- Shape primitives of the minimal set are constructed [38]
- The resulting shape primitives are tested against the entire dataset to determine the score of the shape (Note: the score of the shape measures how well shape primitives match with points in the data) [38]
- The shape with the highest score of shape is extracted [38]
- Algorithm moves onto other random points in the data and carries out the steps again [38]

```
 $\Psi \leftarrow \emptyset$  {extracted shapes}

 $\mathcal{C} \leftarrow \emptyset$  {shape candidates}

repeat

     $\mathcal{C} \leftarrow \mathcal{C} \cup \text{newCandidates}()$  {see sec. 4.1 and 4.3}

     $m \leftarrow \text{bestCandidate}(\mathcal{C})$  {see sec. 4.4}

    if  $P(|m|, |\mathcal{C}|) > p_t$  then

         $\mathcal{P} \leftarrow \mathcal{P} \setminus \mathcal{P}_m$  {remove points}

         $\Psi \leftarrow \Psi \cup m$ 

         $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathcal{C}_m$  {remove invalid candidates}

    end if

until  $P(\tau, |\mathcal{C}|) > p_t$ 

return  $\Psi$ 
```

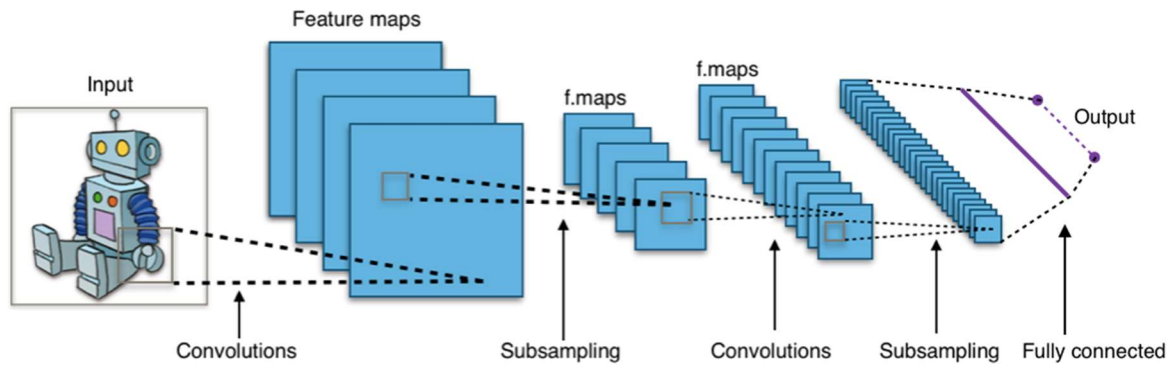
**Figure 27.** RANSAC Algorithm [38]

With all these tools its possible to build a 3D object detection pipeline, from data processing , training and testing, all the way to the output of a bounded box around the detected object.

### 2.2.2 Deep learning approach: image based processing method

There are three main types of neural layers in a Convolutional Neural Network (CNN) [42]:-

- Convolutional layers
- Pooling layers
- Fully connected layers



**Figure 28.** Generic CNN [42]

Feature maps from input data are generated by the convolutional layer. The spatial dimensions of these maps are “flattened” by the pooling layers and the outputs are fed into the fully connected layers which outputs a 1d feature vector for classification or object proposals and bounding box predictions in the case of object detection [42]

In M M Rahman et al research paper they state that this method uses strictly RGB images as input modality [11]. It uses hand crafted features or basic 2D object detectors to create 2D region proposals and re-projection or regression from 2D bounding boxes to generate 3D bounding boxes [11]. An example of this method was introduced by Chen et al in their paper called “*3D Object Proposals for Accurate Object Class Detection*” [21]. The pipeline of this model goes as follows [21] :-

- Take in stereo images as input modality
- Generates depth using stereo images
- Uses the Markov Random Field (MRF) to generate object proposals as 3D bounding boxes.
- Re-project these 3D bounding boxes as 2D bounding boxes on an RGB image.
- Feed the resulting image to a Fast R-CNN network for classification

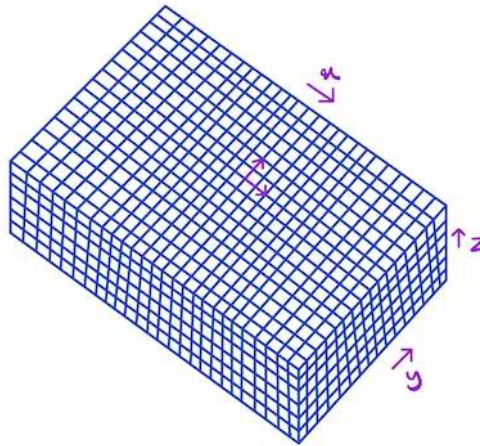
The research highlighted a lack of depth information as a limitation which is imperative to the systems accuracy in terms of the scale of the object and its localization [11]. Also if this method is implemented using a region proposal network, it may do poorly with occluded and truncated objects [11].

### 2.2.3 Point cloud based processing methods

This method can be grouped into 3 categories namely

**View-Based:** This method takes point clouds as its input modality. This is a 2D driven method in that it converts the point cloud to 2D image views for processing and then deriving its 3D bounding boxes using 2D bounding boxes generated from typical CNN architecture [11].

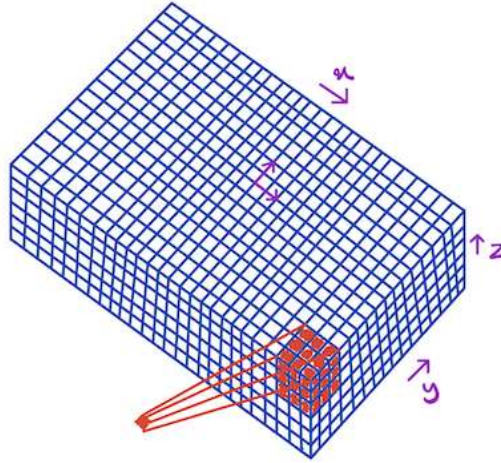
**Voxel-Grid Based:** This method is captured in Y, Zhou et al's paper, "*VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*" [22]. Unlike image based processing methods, this method doesn't need features to be hand-crafted, instead features are extracted from the point clouds and bounding boxes are generated at the same time.



**Figure 29.** Illustration of 3D Voxelization [10]

The VoxelNet architecture is made up of 3 parts [22]:-

- **Feature learning network:** In this block, point clouds are divided into groups of 3D voxels. The voxels are further processed by way of random sampling to avoid sampling bias by increasing the balance of points between voxels, this strategy also leads to computational savings [22]. This block also contains a series of stacked Voxel Feature Encoding (VFE) layers which transforms the set of points in the 3D voxel to a unified feature representation [22]. These feature representations are then represented as a sparse 4D tensor [22].
- **Convolutional Middle Layers:** This block takes the output from the feature learning network and performs 3D convolution followed by applying the batch normal layer and the ReLU layer in that order [22]. This process crystalizes the features obtained and preserves the shape description.



**Figure 30.** Illustration of 3D Convolution [10].

- **Regional Proposal Network:** As a refresher, this is a two stage detection network unlike **Single Shot detection (SSD) networks**. These networks use explicit region proposal networks instead of dense prediction paradigm as with the case of SSD networks [10]. This block takes the output from the Convolutional Middle Layer as input and generates feature maps [22].

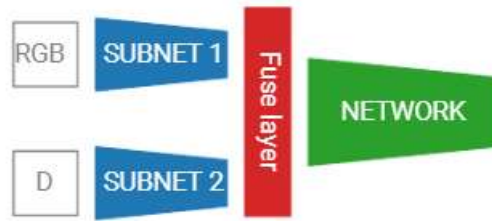
**Unstructured point cloud based:** unlike voxel grid based methods this method handles unordered point clouds directly without any loss of information occurring. An example of this method is PointNet [2]. which was mentioned earlier in section 2.1.2.

#### 2.2.4 Multimodal Fusion-based

This method fuses the previous mentioned methods, image-based and point clouds to form one method that provides texture information and depth information together as its output [11]. An example of this method was explored by T, Ophoff et al in their paper, “*Exploring RGB+Depth Fusion for Real-Time Object Detection*” [23]. Where they concluded that fusing both image and depth information could lead to better performance.

##### Network Architecture

The architecture of the network they proposed comprised 2 subnetworks, one for both RGB and depth input modalities. The features maps derived from both subnetworks are then concatenated in a fuse layer and then sent to an SSD network (e.g. YOLO detector) [23].



**Figure 31.** Multimodal Fusion based Network Architecture [23]

It is important to note that The fuse layer divides output channels by two and performs a 1 x 1 convolution on the concatenated output. This operation flattens the feature maps generated to the size of one network. This creates a more robust combined feature map than feature maps from both subnetworks individually [23].

### 2.3. Choice of methods

After weighing all the pros and cons with the different approaches to the 3d object detection problem discussed in this chapter, point cloud based deep learning approach was selected. View-based lacked depth information and when depth information was provided, it was subject to loss of information when projecting to a 2D image plane, voxel grid based has efficiency issues with data handling resulting in higher computational cost.

Initially, I started with the ROS and PCL libraries solution principle to solve the project aim because of the seamless compatibility between the xtionPRO live RGB-D camera (which the option was available to introduce it into the project to be used for real time object detection if time permitted) and the ROS framework. However access to technologies like CUDA for GPU, for faster processing, was lost due to the lock down because the laptop available kept crashing every time the camera was turned on for a while, while using the virtual machine. To solve this issue, Google Collab was used because of their free GPU. A deep learning approach was decided upon where I could use Pytorch to implement deep neural networks.

Pytorch is based on the torch library, it is a machine learning framework available to the community via open source [24]. Together with OpenCV, Python technologies I decided to implement the VoteNet architecture introduced by C, Qi et al in their research paper, "*Deep Hough Voting for 3D object Detection in Point Clouds*" [1]. The network produces state of the art results without a computational cost trade-off, using only point clouds as its input modality [1] on the SUN-RGB-D and ScanNet datasets.

The VoteNet network takes uses of PointNet++[25], an improved version of PointNet [2] as its backbone network for handling unordered point clouds[2], making it less susceptible to information loss and is efficient in dealing with sparsity in point clouds by only focusing on perceived points [1]. Unlike voxel based approaches that transform point clouds into rigid structures, VoteNet can handle point clouds raw [1] and doesn't rely 2D detectors like image view based approaches. It utilizes Hough voting to generate 3D box proposals. This solves the problem most other point based methods have which is the absence of points in 3D object centroids, which makes it difficult for point based methods to understand a scene close to object centroids [1].

**Backbone Networks:** backbone networks are feature encoders for input modalities [10]. The idea with backbone networks is to use part of a different network in another network. For example part of ImageNet can be used as a backbone network for an object detection network [10] and in this project PointNet++ was used as a backbone for VoteNet [1]. Other Popular backbone networks are AlexNet, VGG, ResNet [10].

**Deep Hough Voting:** the idea of Hough voting in 2d detection works as such, a codebook is used to store feature maps and their distance to object centroids. Interest points are then sampled for feature extraction around them [1]. These features are then compared to the features in the codebook and votes are computed, distances between the point and object centers are relayed [1]. Clusters of points will form in the middle of the target object and this makes it easier to generate bounding boxes around the target object. In Deep hough voting for 3Ddetection, interest points are determined by deep neural networks and not by hand crafted features [1] and the network is used and not codebook to generate votes. The overall principles are still the same.

## **Chapter 3**

### **Software Requirements and System Design**

#### **Section 3.1 Software Requirements (or another appropriate title)**

This section describes the requirements of the target system. Since this was an exploratory software the requirements would be generic because the project is exploring.

Figuring out the requirements took a 3 step process as follows [25] :-

- Carrying out feasibility study
- Requirements were gathered
- Requirements were made official

##### **Carrying Out Feasibility Study:**

After the project topic was chosen, I was advised by my project supervisor to embark on a solution that was feasible and achievable before the deadline. And it was understood that if there was adequate time I could choose to implement my solution to 3D object detection in real time using the RGB-D camera, xtionPRO live camera. Unfortunately I couldn't achieve that optional requirement because of lack of access to school due to the lockdown, I was forced to change my method which made me lose time.

##### **Requirements were gathered:**

For this step, I held meetings with my project supervisor and he gave great input on what requirements would be necessary and adequate. Requirements discussed were on data types, amount of objects it can track per frame, real time detection and occlusion.

##### **Requirements were made official:**

- Input modality of the system will be point clouds
- System will detect objects from the point clouds
- System will be able to detect multiple objects
- System will put bounding boxes around target objects

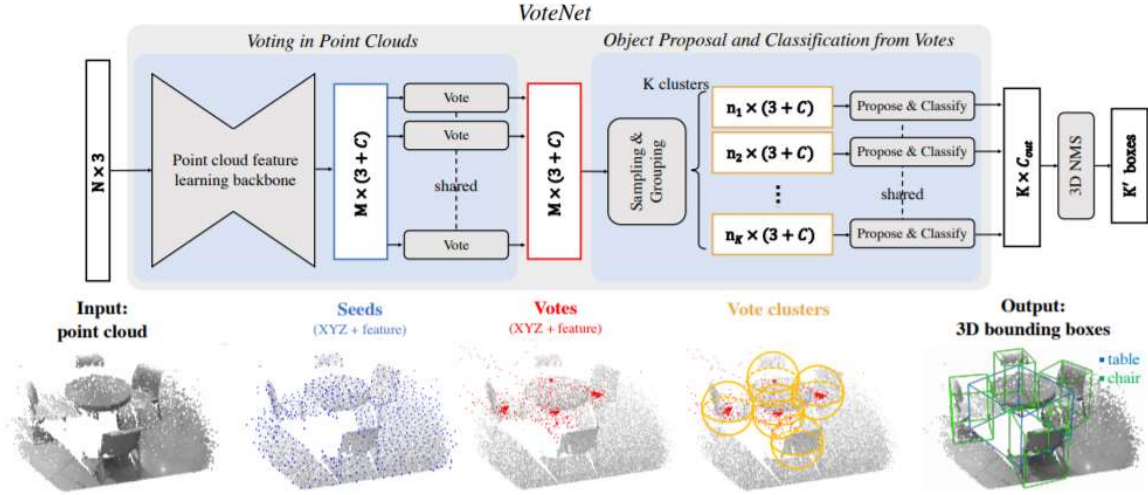
#### **Section 3.2. System Design**

This section will focus on the technologies used to build the system and overall design of the system which implements the VoteNet architecture [1] and the changes made to the architecture and code. Since this was a project about exploratory software, freedom was given to choose what programming languages, libraries, tools, datasets and solution principles (stated in section 2.2) to solve the 3D object detection problem.



### 3.2.1 System Architecture

The architecture has two blocks, one block for generating votes from point cloud data and the other block works on the votes for object proposal and classification [1].



**Figure 32.** Illustration of VoteNet Architecture [1]

#### Generating Votes

This block generates votes (i.e. virtual points) from a point cloud with  $N(x,y,z) \times 3$  size, where  $x,y,z$  are the 3D coordinates [1]. The votes all have 3D coordinates and a high dimensional feature vector [1]. The backbone network, is based on PointNet++. It has 4 set abstraction layers and 2 up-sampling layers [1].

The set abstraction layers have receptive radius of 0.2, 0.4, 0.8 and 1.2. and the inputs are sub-sampled to 2048, 1024, 512, and 256 points [1]. The 4<sup>th</sup> layer is up-sampled by the up-sampling layers using 3D coordinates and 256-dim features [1].

So the backbone network undertakes the feature learning process from the point cloud data and outputs points with 3D coordinates. These points are the inputs for the deep hough voting phase where seed points are learned [1]. Each seed points corresponds to one vote [1]. The voting module is accomplished using a multi-layer perceptron (MLP) with fully connected layer output sizes of 256, 256, 259, ReLU, batch normalization (BN)[1] and an extra convolution layer. The extra convolution layer doesn't exist on the original VoteNet, it is a new addition to explore the architecture to see if it has any impact on accuracy.

```
15
16 class VotingModule(nn.Module):
17     def __init__(self, vote_factor, seed_feature_dim):
18         """ Votes generation from seed point features.
19
20         Args:
21             vote_facotr: int
22                 number of votes generated from each seed point
23             seed_feature_dim: int
24                 number of channels of seed point features
25             vote_feature_dim: int
26                 number of channels of vote features
27         """
28         super().__init__()
29         self.vote_factor = vote_factor
30         self.in_dim = seed_feature_dim
31         self.out_dim = self.in_dim # due to residual feature, in_dim has to be == out_dim
32         self.conv1 = torch.nn.Conv1d(self.in_dim, self.in_dim, 1)
33         self.conv2 = torch.nn.Conv1d(self.in_dim, self.in_dim, 1)
34         self.conv3 = torch.nn.Conv1d(self.in_dim, self.in_dim, 1)
35         self.conv4 = torch.nn.Conv1d(self.in_dim, (3+self.out_dim) * self.vote_factor, 1)
36         self.bn1 = torch.nn.BatchNorm1d(self.in_dim)
37         self.bn2 = torch.nn.BatchNorm1d(self.in_dim)
38
```

**Figure 33.** Code Snippet of Voting\_module.py [1]

The features generated by the backbone network is fed into the MLP which outputs:-

- Euclidean space offset  $\Delta \mathbf{x}_i \in \mathbf{R}^3$
- Feature offset  $\Delta \mathbf{f}_i \in \mathbf{R}^c$

The votes generated from the seed points  $\{\mathbf{s}_i\}_{i=1}^M$  has  $\mathbf{y}_i = \mathbf{x}_i + \Delta \mathbf{x}_i$  and  $\mathbf{g}_i = \mathbf{f}_i + \Delta \mathbf{f}_i$  [1]. The regression loss supervises the predicted 3D offset  $\Delta \mathbf{x}_i$ , explicitly [1].

### Classification and Object proposal

The votes are used for context aggregation and clustered through uniform sampling and grouping with respect to spatial distance so that their features can be aggregated using a shared PointNet for object proposals and classification [1].

More changes to the original VoteNet implementation will be discussed in chapter 4

### 3.2.2 Technologies Used

This sub section will discuss the technologies used to build this system so that it can be replicated or used as a foundation for future work. The technologies used would be categorized into:-

- Data Processing
- Training and Evaluating Model
- Output Visualization

#### Data processing

Technology	Version	Use
OpenCV	Current version	For Computer vision operations
Python Programming Language	3.0	Build the code
Numpy Arrays	1.8	Data manipulation
Google Drive		To store SUN RGB-D data
MATLAB	Current version	To extract point clouds and annotations from SUN RGB-D dataset

**Table 1.** Technologies used for data processing

#### Training and Evaluating Model

Torchvision	0.4	Package that contains model architecture for computer vision
CUDA	10	GPU processing
Google Collab		For free GPU
couchDB		Object database for ROS with linemod implementation
Object Recognition Kitchen (ORK)		Object detection using ROS

**Table 2.** Technologies used for evaluating model

### Output Visualization

Meshlab	Meshlab 2020.07	To vizualize the point clouds and the predicted bounding boxes
ROS, rviz	noetic	To visualize and record stream of point clouds

**Table 3.** Technologies used for visualization.

## Chapter 4

### Software Implementation

#### 4.1 ROS Implementation of object detection using linemod

The following is a step by step implementation of object detection using ROS, ORK, CouchDB and linemod based on this tutorial [43].

- After installing and setting up the ROS working environment.
- ORK software was installed [43]
- Rqt\_reconfigure and Rviz were also installed using this command

```
o sudo apt-get install ros-<your ROS noetic>-rviz ros-<your ROS noetic>-  
rqt_reconfigure ros-<your ROS noetic>-openni2
```

- The launch driver for the Asus XtionPRO live camera was launched using this command [43]

```
o roslaunch oppenni2_launch oppenni2.launch
```

- RViz was ran using this command [43]

```
o rosrune rviz rviz
```

- in the Rviz graphic user interface, */camera/driver topic* was selected and depth registration was enabled
- A point cloud topic was added to */camera/depth/points*
- Color transformer was set to RGBB so that color and 3D points were visible [43]
- The object detection database was setup using CouchDB database
- Objects and mesh were uploaded to the database, the ORK already has a few objects and meshes [43].

```
o rosrune object_recognition_core object_add.py -n "coke " -d "coke"  
o rosrune object_recognition_core mesh_add.py <YOUR_OBJECT_ID> <path  
to ork_tutorials/data/coke.stl>
```

- features from objects can be learnt from the database by configuring and executing linemod in training mode [43].

```
o rosrune object_recognition_core training -c `rospack find  
object_recognition_linemod`/conf/training.ork
```

- after training, linemod was configured to define a source, pipeline and sink and was executed in detection mode

```
o rosrune object_recognition_core detection -c `rospack find  
object_recognition_linemod`/conf/detection.ros.ork
```

- to visualize the process and see the output, orkobject was added as a display and the necessary topics were also added, namely, object id, name etc.

Unfortunately I wasn't able to replicate the tutorials [43] output.

## 4.2 VoteNet Implementation

The VoteNet was implemented using Google Colab to train, test and evaluate the model.

This section will be broken down into four parts.

- SUN RGB-D Dataset preparation
- Training VoteNet based model
- Implement VoteNet based model
- Input stream of point clouds from RGB-D camera using ROS

### 4.2.1 SUN RGB-D Dataset Preparation

The SUN RGB-D data was gotten from the SUN RGB-D website [30]. It has 10,355 RGB-D images that contain RGB-D images from NYU depth v2 [27], Berkeley B3DO [28], AND SUN3D [29] images. The images come with high quality 2D and 3D dense annotations [31] and contains 146,617 2D polygons and 64,595 3D bounding boxes with their corresponding orientations. It is a PASCAL-scale RGB-D benchmark suite that's perfect for training deep learning models for indoor scene understanding [31].



**Figure 34.** Images coupled with their annotations of SUN RGB-D Dataset [31].

The dataset was captured using 4 different RGB-D cameras [31]:-

- Intel RealSense
- Asus Xtion

- Kinect v1
- Kinect v2

And because of its scale and diversity, models trained on it can avoid overfitting as it would have been the case if the data size was small [31]. This dataset was chosen as part of my plan to detect objects in an indoor scene.

To prepare the SUN RGB-D data for VoteNet, point clouds and annotations can be extracted using matlab but this project used an already pre-processed data gotten from github [45] and the code

```
!python sunrgbd_data.py -gen_v1_data
```

was ran to prepare the data for training on Google Colab.

These annotations include class, v2, 2D –xmin, ymin, xmax,ymax, and 3D bounding boxes, centroids, size, 2D heading [33].

#### **4.2.2 Training VoteNet based model**

Dataset for deep learning usually have training dataset, validation dataset and test dataset [34]. The model is first trained using training dataset along with an optimizer, in this case, the Adam optimizer is used, this method is called supervised learning [34]. It provides great results relatively fast compared to other optimization methods [35]. Alpha, beta1, beta2 and epsilon are the parameters for Adam optimizer [35].

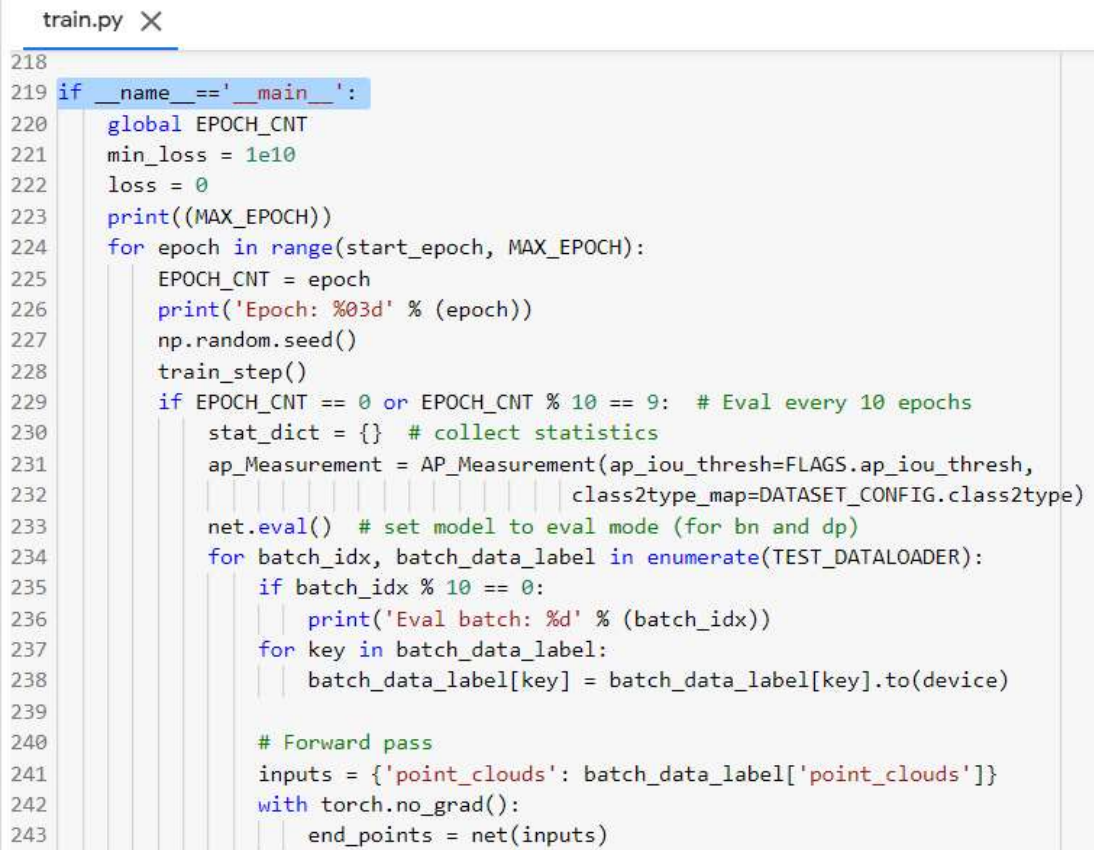
- Alpha: Is the learning rate
- Beta1: is the first moment estimates exponential decay rate
- Beta2: is the second moment estimates exponential decay rate
- Epsilon: is the smallest number that prevents any division by zero [35]

Adam handles sparse gradients by combining the advantages of both AdaGrad and RMSProp algorithms [35].

The weights of the neural networks are learnt and adjusted during the training process via back propagation. The validation dataset is used to evaluate the trained model while fine tuning the models hyperparameters and the test dataset is used on the final model. The SUN RGB-D dataset was split into only training and validation datasets [34]. The Votenet model was trained using approximately 5000 RGB-D training images and approximately 5000 validation images annotated with their corresponding 3D bounding boxes



The majority of the code for this project comes from the official github repository of QI et al's paper [33]. Changes were made to train.py, /models/vote\_module.py, /models/ap\_helper.py. The original train.py file needed to call train.py after every epoch but this project integrated that into the main loop of train.py, so that it trains for every epoch(180) with one function call.



```
train.py X
218
219 if __name__ == '__main__':
220     global EPOCH_CNT
221     min_loss = 1e10
222     loss = 0
223     print((MAX_EPOCH))
224     for epoch in range(start_epoch, MAX_EPOCH):
225         EPOCH_CNT = epoch
226         print('Epoch: %03d' % (epoch))
227         np.random.seed()
228         train_step()
229         if EPOCH_CNT == 0 or EPOCH_CNT % 10 == 9: # Eval every 10 epochs
230             stat_dict = {} # collect statistics
231             ap_Measurement = AP_Measurement(ap_iou_thresh=FLAGS.ap_iou_thresh,
232                                             class2type_map=DATASET_CONFIG.class2type)
233             net.eval() # set model to eval mode (for bn and dp)
234             for batch_idx, batch_data_label in enumerate(TEST_DATA_LOADER):
235                 if batch_idx % 10 == 0:
236                     print('Eval batch: %d' % (batch_idx))
237                     for key in batch_data_label:
238                         batch_data_label[key] = batch_data_label[key].to(device)
239
240                     # Forward pass
241                     inputs = {'point_clouds': batch_data_label['point_clouds']}
242                     with torch.no_grad():
243                         end_points = net(inputs)
```

**Figure 35.** Code snippet of Train.py [1]

( Note that changes to vote\_module.py was discussed in section 3.2.1).

The change in test\_run.py , a for loop iterating through point clouds allows the network take stream of point clouds.



```
test_run.py X
57 optimizer = optim.Adam(net.parameters(), lr=0.001)
58 checkpoint = torch.load(checkpoint_path)
59 net.load_state_dict(checkpoint['model_state_dict'])
60 optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
61 epoch = checkpoint['epoch']
62 print("Loaded checkpoint %s"%(checkpoint_path))
63
64
65 for pc_path in os.listdir(path_to_point_clouds):
66     # Load and preprocess input point cloud
67     net.eval() # set model to eval mode (for bn and dp)
68     point_cloud = read_ply(path_to_point_clouds+'/'+pc_path)
69     pc = preprocess_point_cloud(point_cloud)
70     print('Loaded point cloud data: %s'%(pc_path))
71
72     # Model inference
73     inputs = {'point_clouds': torch.from_numpy(pc).to(device)}
74     tic = time.time()
75     with torch.no_grad():
76         end_points = net(inputs)
77     toc = time.time()
78     print('Inference time: %f'%(toc-tic))
79     end_points['point_clouds'] = inputs['point_clouds']
80     pred_map_cls = parse_predictions(end_points, eval_config_dict)
81     # print(pred_map_cls)
82     print('Finished detection. %d object detected.'%(len(pred_map_cls[0])))
83     dataset='sunrgbd'
```

**Figure 36.** Code snippet of test\_run.py [1]

In /models/ap\_helper.py faster non maximum suppression is used to remove overlapping boxes [44]

```
ap_helper.py X
18 from box_util import get_3d_box
19 sys.path.append(os.path.join(ROOT_DIR, 'sunrgbd'))
20 from sunrgbd_utils import extract_pc_in_box3d
21 def nms_3d_faster(boxes, overlap_threshold, old_type=False):
22     x1 = boxes[:,0]
23     y1 = boxes[:,1]
24     z1 = boxes[:,2]
25     x2 = boxes[:,3]
26     y2 = boxes[:,4]
27     z2 = boxes[:,5]
28     score = boxes[:,6]
29     area = (x2-x1)*(y2-y1)*(z2-z1)
30     I = np.argsort(score)
31     pick = []
```

**Figure 37.** Code snippet of ap\_helper.py [1][44]

As mentioned in technologies used, VoteNet is compatible with CUDA 10. So the CUDA version on Google Collab was uninstalled and CUDA 10 was installed. Following that, CUDA layers were compiled for the backbone network, PointNet++, on Google Collab using the code below

***!python setup.py install***

The SUN RGB-D dataset were saved in google drive and mounted on Google Collab and was used to train the network. The VoteNet network was trained using an Adam optimizer with a learning rate of 0.001 and after 80 and 120 epochs, it decreases by 10x [1]. VoteNet was trained on Google Collab using the code below

***!python train.py --log\_dir log***

The weights of the neural network are saved in checkpoint-sunrgbd.Tar.

The network in one forward pass has the ability to generate proposals from input point clouds [1]. Using the pre-processed data, the trained model is saved for testing and evaluation. More on testing and evaluation in section 5.

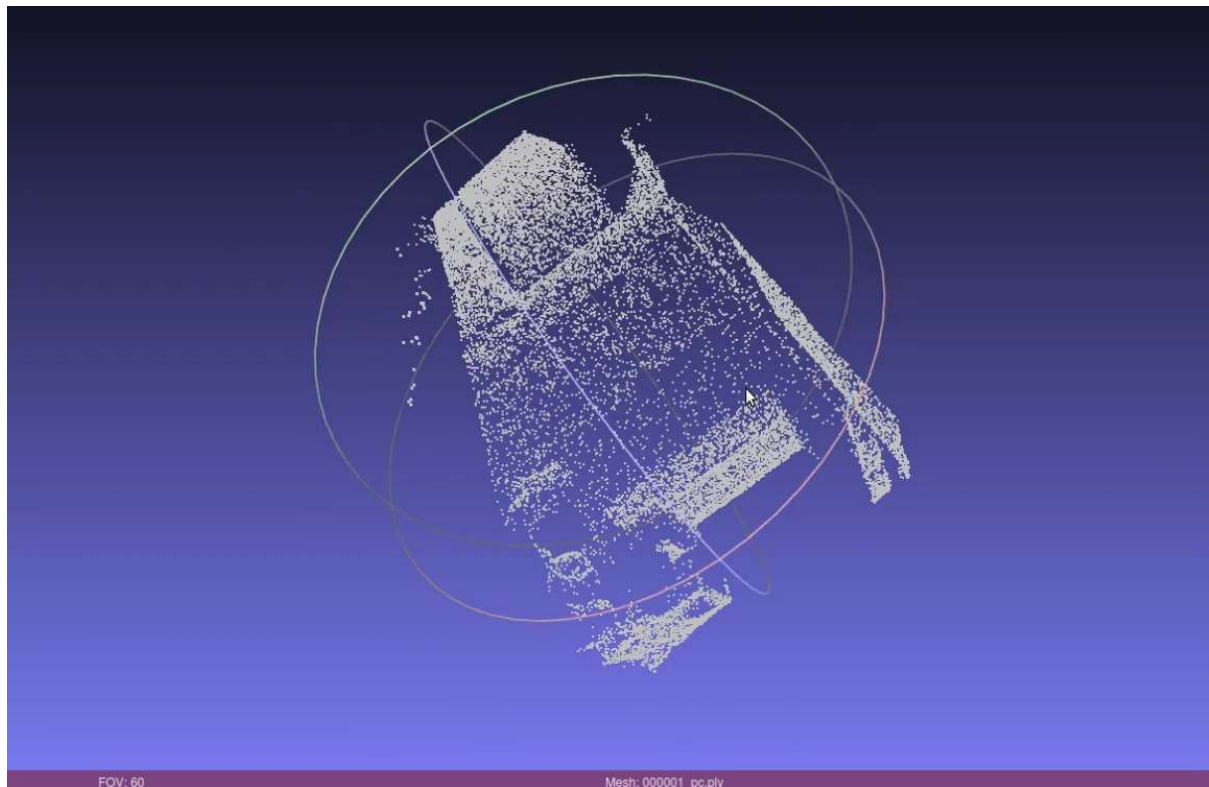
After the model has been trained it is evaluated and tested using SUN RGB-D data, the results of 3D detection would be placed in demo\_files/sunrgbd\_results directory for test results and eval\_sunrgbd folder contains results on validation data.

To visualize the point clouds and predicted bounding boxes, meshlab was used. Meshlab is 3D editing and modelling tool, it is good for handling point clouds and meshes. it is an open source system [32].

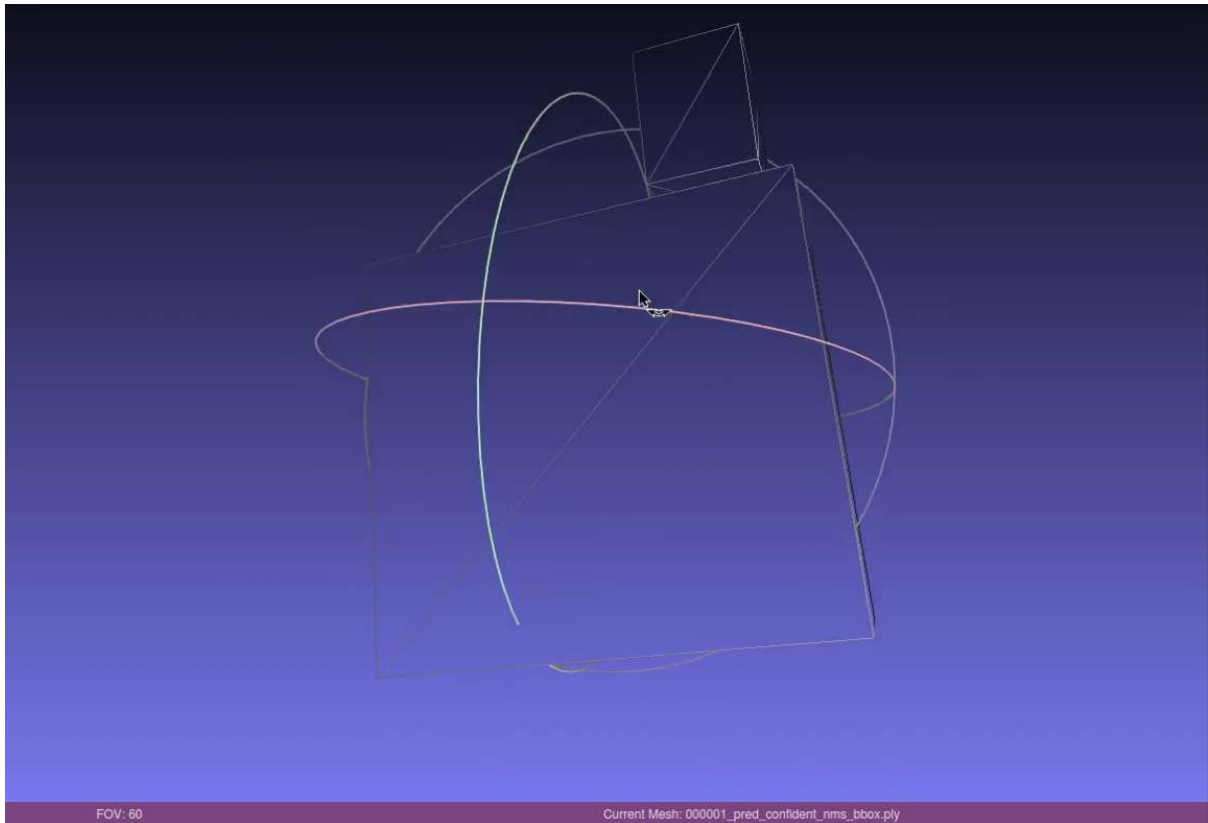
Three files were taken from the results directory

- \*\_pc.ply (It contains point clouds of input )
- \*\_pred\_confident\_nms\_bbox.ply(It contains predicted bounding boxes)
- \*\_pred\_map\_cls.txt (It contains information of class the output belongs to)

So to visualize the point clouds and bounding boxes on meshlab these three files of the same names were opened on meshlab.



**Figure 38.** Point clouds of 000001pc.ply input file



**Figure 39.** Results with predicted bounding boxes of 000001pc.ply

Classification of the objects are written in pred\_map\_cls.txt. The network is trained for 10 classes out of the 37 classes available in SUN RGB-D.

Suppose in \*\_pred\_map\_cls.txt, it is written that

```
000001_pred_map_cls - Notepad
File Edit Format View Help
3 0.6531675584173825,0.599676073633194,1.9259330415387255,0.6212762121
3 0.7509084883138295,0.5275864201710224,1.410589969232032,0.6945360310
1 0.6131452064797643,0.4665589046869278,2.358961032499279,1.1419884196
3 0.9285063249044868,0.6221758584604263,1.4606888345614926,0.479902949
4 -0.19585013187621908,0.5087954059243203,1.9068078012083323,-0.573926
```

**Figure 40.** 000001\_pred\_map\_cls

here, 3,3,1,3,4 are class names from

['bed','table','sofa','chair','toilet','desk','dresser','night\_stand','bookshelf','bathtub'] list in the /sunrgb/sunrgb\_data.py file. This means that 3 is a chair, 4 is toilet and 1 is a table. So the system is predicting that there is a chair, table and/or toilet in the point cloud.

### **Input stream of point clouds from RGB-D camera using ROS**

The model of the camera used that was supposed to be used was Asus xtionPRO live. The idea was that the camera would capture point clouds in real time using the robot operating system (ROS) and the points would be used in Google Collab by the VoteNet model to detect objects.

### **Capturing point clouds using RGB-D camera**

After carefully installing ROS noetic openni2 for Asus was installed using the following command [46]

```
sudo apt-get install ros-noetic-rgbd-launch ros-noetic-openni2-camera ros-noetic-openni2-launch
```

roscore was ran in a new terminal using the command [46]

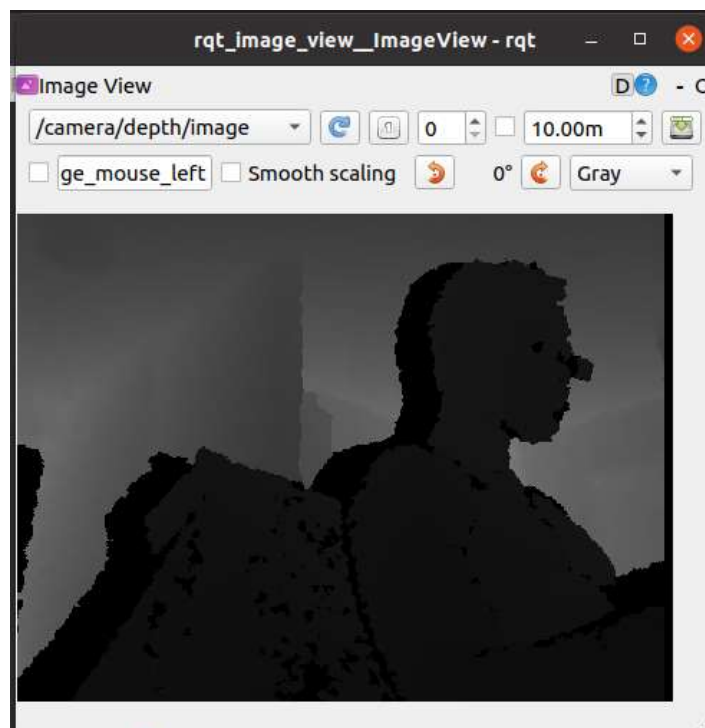
```
roscore
```

then the following command was ran in another new tab to launch openni2 for Asus [46]

```
roslaunch openni2_launch openni2.launch
```

to visualize the depth image from the camera using rqt the following command was ran [46]

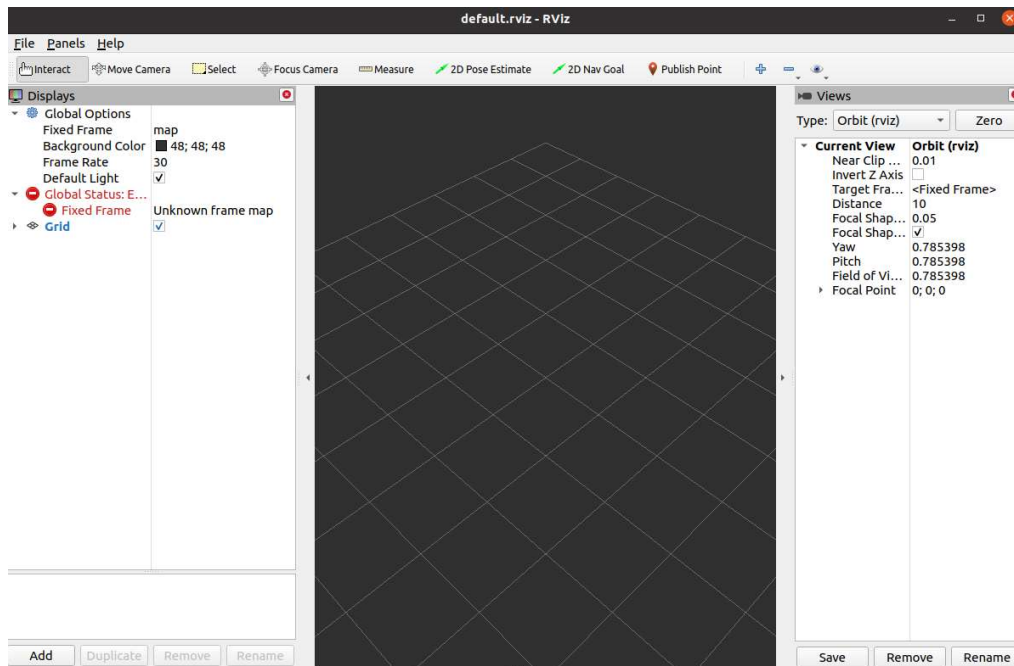
```
roslaunch rqt_image_view rqt_image_view
```



**Figure 41.** xtionPRO live depth images on rqt

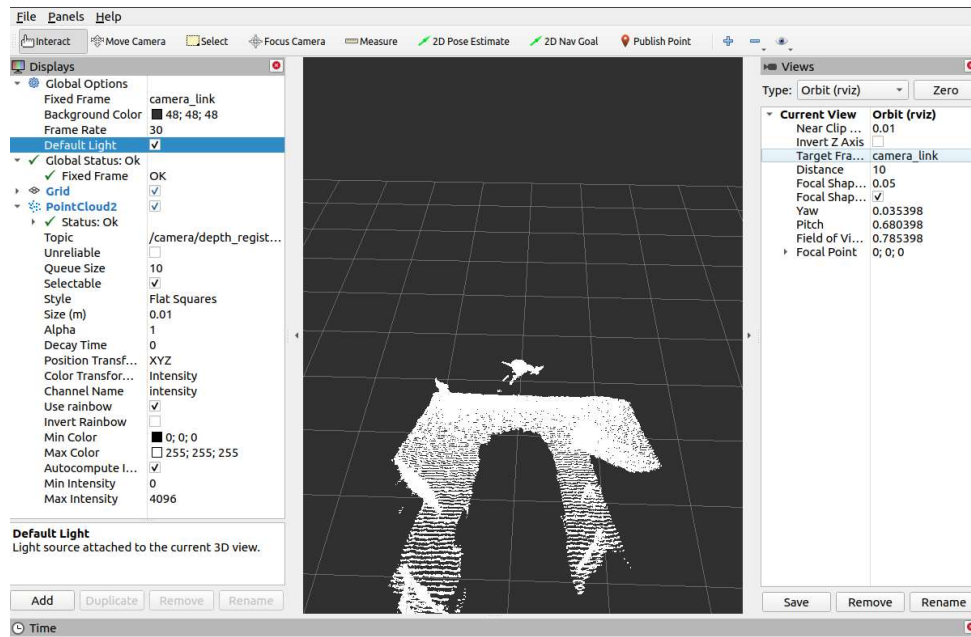
To visualize the point clouds generated by the RGB-D camera, Rviz was used. It is a GUI for robot visualization tool that helps users visualize point clouds from sensors and developing robots. We run rviz using the following command [46]

***roslaunch rviz rviz***



**Figure 42.** RViz

Then the point clouds sensors are added to rviz



**Figure 43.** Point clouds on RViz

the streams of points clouds can be recorded using rosbag and saved to run later on the model.



## Chapter 5

### Software Testing and Evaluation

#### 5.1 Testing

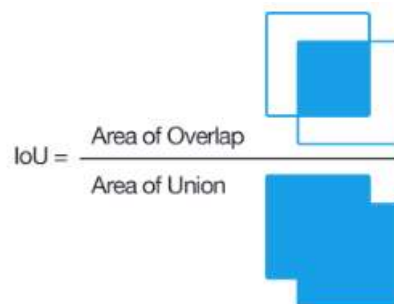
VoteNet model was tested and evaluated with the following command

```
!python eval.py
```

This evaluated the model and printed the results on Google Collab. The results are saved in *eval\_sunrgbdb/log\_eval.txt*

#### 5.2 Evaluation

VoteNet performance as stated earlier is state of the art. The network was evaluated using the Average Precision with a 3D IOU threshold of 0.25. IOU is an evaluation metric for object detectors on a given dataset [36] which means intersection over union. The dataset needs to include ground truth bounding boxes and predicted bounding boxes so that IOU can be applied [36].


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

**Figure 44.** IOU equation [36]

Top refers to Area of overlap and the bottom refers to area of union

Average precision and average recall are also metrics used to determine how accurate the model is. Precision is the accuracy of the predicated bounding boxes and recall measures the rate at which the right predictions are made [36]. This means that mean average precision(mAP) is the average precision(AP) for all the classes and average recall(AR) is the average recall for all the classes [36].

$$\text{Precision} = \frac{tp}{tp + fp}$$



$$\text{Recall} = \frac{tp}{tp + fn}$$

Where [36] tp = true positive

Fp = false positive

Fn = false negative

### 5.3 Results

Results for average precision at an iou\_thresh: 0.250000. The model had a mAP: 0.550887 and average recall: 0.702549

Class	Precision	Recall
Bed	0.801268	0.836096
Table	0.432034	0.638855
Sofa	0.604177	0.719875
Chair	0.738769	0.841681
Toilet	0.890247	0.94702
Desk	0.156603	0.268077
Dresser	0.206317	0.541667
Night Stand	0.601590	0.787402
Bookshelf	0.281861	0.598662
Bathtub	0.796007	0.846154

**Table 4.** Results for IOU threshold 0.25

Results for Recall at an iou\_thresh: 0.50000. the model had an mAP of 0.314628 and average recall of 0.442529

Class	Precision	Recall
Bed	0.447454	0.569061
Table	0.156831	0.326789
Sofa	0.393924	0.528951
Chair	0.526565	0.64398
Toilet	0.592113	0.715232
Desk	0.030086	0.107231
Dresser	0.101566	0.319444
Night Stand	0.353434	0.531496
Bookshelf	0.045972	0.16388
Bathtub	0.498336	0.519231

**Table 5.** Results for IOU threshold 0.5

The model trained for this project did slightly worse compared to the results gotten by Qi et al's [1] implementation on github [33].

This model had a mAP of 55 while the Qi et al's [33] results were 57 for an IOU threshold of 0.25. for a IOU threshold of 0.5 the results were virtually identical, this projects model had a score 31 while Qi et al's model had a score of 32. The extra convolutional layer added to the voting module had little to no effect on the results.

## Chapter 6

### 6.1 Conclusions

The aim of the project was to perform a deep learning approach for RGB-D object detection, to detect indoor objects using the SUN RGB-D benchmark dataset. A literature review was performed on related or previous solutions to 3D object detection problem. This paper looked at recent advancements in 3D technology and focused on Qi et al's end to end solution for handling raw unordered point clouds, PointNet, which was a backbone network for this project's deep learning solution.

Two different approaches were attempted. The approach on the ROS couldn't be investigated further because of the lack of technologies available but the concept for creating an object detection pipeline was learnt and understood.

The deep learning approach achieved the aim of the project partially. Objects in point clouds were detected using 3D bounding boxes but unfortunately it was only tested on validation data and not data captured by me using the xtionPRO live camera. The virtual machine kept crashing on my working environment everytime the camera was plugged in. More time was needed to work around this issue on the technologies available. Training the data was very challenging on Google Colab because of the data size and the GPU available but judging by the results of the trained model, it was trained properly as its mAP scores were close to the official implementation done by Qi et al in [33]

Regardless, the idea of how to implement a deep learning approach was learnt, and after investigating all the different methods available for this task, knowing which method to employ for certain situations was learnt.

### 6.2 Further Research

As mentioned by Rahman et al [11] in their survey of advances in 3D object detection there are lots of exciting opportunities for further research in areas like occlusion, data handling, multi-modality fusion etc, but for this specific project, real time 3D object detection would be an area one can build on, as this was not achieved by this project. One can go further and deploy the object detection system in other portable devices like cell phones, adding smart features like distance measurement, counting number of objects in a scene.

## List of References

1. C. Qi, O. Litany, K. He and L. Guibas, "Deep Hough Voting for 3D Object Detection in Point Clouds", *Arxiv.org*, 2020. [Online]. Available: <https://arxiv.org/pdf/1904.09664.pdf>. [Accessed: 29- Aug- 2020].
2. Qi, C., Su, H., Mo, K. and Guibas, L. 2020. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *Arxiv.org*. [Online]. [Accessed 29 August 2020]. Available from: <https://arxiv.org/pdf/1612.00593.pdf>.
3. Xiang, Y., Schmidt, T., Narayanan, V. and Fox, D. 2020. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *Arxiv.org*. [Online]. [Accessed 29 August 2020]. Available from: <https://arxiv.org/pdf/1711.00199.pdf>.
4. Dang, X., Kang, B., Liu, X. and Cui, G. 2020. An Interactive Care System Based on a Depth Image and EEG for Aged Patients with Dementia. *researchgate.net*. [Online]. [Accessed 29 August 2020]. Available from: [https://www.researchgate.net/publication/318540109\\_An\\_Interactive\\_Care\\_System\\_Based\\_on\\_a\\_Depth\\_Image\\_and\\_EEG\\_for\\_Aged\\_Patients\\_with\\_Dementia](https://www.researchgate.net/publication/318540109_An_Interactive_Care_System_Based_on_a_Depth_Image_and_EEG_for_Aged_Patients_with_Dementia).
5. Macnish, K. 2020. Surveillance Ethics | Internet Encyclopedia of Philosophy. *iep.utm.edu*. [Online]. [Accessed 29 August 2020]. Available from: <https://iep.utm.edu/surv-eth/>.
6. Lauronen, M. 2020. ETHICAL ISSUES IN TOPICAL COMPUTER VISION APPLICATIONS. *Jyx.jyu.fi*. [Online]. [Accessed 29 August 2020]. Available from: <https://jyx.jyu.fi/bitstream/handle/123456789/55806/1/URN%3ANBN%3Afi%3Aju-201711084167.pdf>.
7. Bechtel, J. 2020. Two Major Concerns about the Ethics of Facial Recognition in Public Safety. *Design World*. [Online]. [Accessed 29 August 2020]. Available from: <https://www.designworldonline.com/two-major-concerns-about-the-ethics-of-facial-recognition-in-public-safety/#:~:text=There%20have%20been%20two%20major,before%20it%20can%20b>



16. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X. and Xiao, J. 2020. 3D ShapeNets: A Deep Representation for Volumetric Shapes. *3dshapenets.cs.princeton.edu*. [Online]. [Accessed 29 August 2020]. Available from: <https://3dshapenets.cs.princeton.edu/paper.pdf>.
17. Aldoma, A., Marton, Z., Tombari, F., Wohlking, W., Potthast, C., Zeisl, B., Rusu, R., Gedikli, S. and Vincze, M. 2020. Tutorial: Point Cloud Library: Three-Dimensional Object Recognition and 6 DOF Pose Estimation - IEEE Journals & Magazine. *ieeexplore.ieee.org*. [Online]. [Accessed 29 August 2020]. Available from: <https://ieeexplore.ieee.org/document/6299166?denied=>.
18. Mahamkali, N. and Vadivel, A. 2020. OpenCV for Computer Vision Applications. *researchgate.net*. [Online]. [Accessed 29 August 2020]. Available from: [https://www.researchgate.net/publication/301590571\\_OpenCV\\_for\\_Computer\\_Vision\\_Applications](https://www.researchgate.net/publication/301590571_OpenCV_for_Computer_Vision_Applications).
19. Hoffmann, J. 2020. ROS-Based Object Recognition Framework. *GitHub*. [Online]. [Accessed 29 August 2020]. Available from: [https://github.com/joffman/ros\\_object\\_recognition/blob/master/docs/thesis.pdf](https://github.com/joffman/ros_object_recognition/blob/master/docs/thesis.pdf).
20. Foote, T., Marder-Eppstein, E. and Meeussen, W. 2020. tf - ROS Wiki. *Wiki.ros.org*. [Online]. [Accessed 29 August 2020]. Available from: <http://wiki.ros.org/tf>.
21. Chen, X., Kundu, K., Zhu, Y., Berneshawi, A., Ma, H., Fidler, S. and Urtasun, R. 2020. 3D Object Proposals for Accurate Object Class Detection. *Xiaozhichen.github.io*. [Online]. [Accessed 29 August 2020]. Available from: <https://xiaozhichen.github.io/papers/nips15chen.pdf>.
22. Zhou, Y. and Tuzel, O. 2020. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. *Arxiv.org*. [Online]. [Accessed 29 August 2020]. Available from: <https://arxiv.org/pdf/1711.06396.pdf%20em%2017/12/2017.pdf>.

23. Ophoff, T., Beeck, K. and Goedeme, T. 2020. Exploring RGB+Depth Fusion for Real-Time Object Detection. *researchgate.net*. [Online]. [Accessed 29 August 2020]. Available from: [https://www.researchgate.net/publication/331228916\\_Exploring\\_RGBDepth\\_Fusion\\_for\\_Real-Time\\_Object\\_Detection](https://www.researchgate.net/publication/331228916_Exploring_RGBDepth_Fusion_for_Real-Time_Object_Detection).
24. PyTorch 2020. PyTorch. *Pytorch.org*. [Online]. [Accessed 29 August 2020]. Available from: <https://pytorch.org/>.
25. Tutorialspoint 2020. Software Requirements - Tutorialspoint. *Tutorialspoint.com*. [Online]. [Accessed 29 August 2020]. Available from: [https://www.tutorialspoint.com/software\\_engineering/software\\_requirements.htm#:~:text=The%20software%20requirements%20are%20description,from%20client's%20point%20of%20view](https://www.tutorialspoint.com/software_engineering/software_requirements.htm#:~:text=The%20software%20requirements%20are%20description,from%20client's%20point%20of%20view).
26. Qi, C., Yi, L., Su, H. and Guibas, L. 2020. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *Arxiv.org*. [Online]. [Accessed 29 August 2020]. Available from: <https://arxiv.org/pdf/1706.02413.pdf>.
27. N. Silberman, D. Hoiem, P. Kohli, R. Fergus. Indoor segmentation and support inference from rgb-d images. In ECCV, 2012.
28. A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. A category-level 3-d object dataset: Putting the kinect to work. In ICCV Workshop on Consumer Depth Cameras for Computer Vision, 2011.
29. J. Xiao, A. Owens, and A. Torralba. SUN3D: A database of big spaces reconstructed using SfM and object labels. In ICCV, 2013
30. Song, S., Lichtenberg, S. and Xiao, J. 2020. SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. *Rgbd.cs.princeton.edu*. [Online]. [Accessed 29 August 2020]. Available from: <https://rgbd.cs.princeton.edu/>.
31. Song, S., Lichtenberg, S. and Xiao, J. 2020. SUN RGB-D: A RGB-D Scene Understanding Benchmark Suite. *Rgbd.cs.princeton.edu*. [Online]. [Accessed 29 August 2020]. Available from: <https://rgbd.cs.princeton.edu/paper.pdf>.

32. P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, G. Ranzuglia  
**MeshLab: an Open-Source Mesh Processing Tool**  
Sixth Eurographics Italian Chapter Conference, page 129-136, 2008
33. Qi, C., Xie, S. and Dalton, N. 2020. facebookresearch/votenet. *GitHub*. [Online].  
[Accessed 29 August 2020]. Available from:  
<https://github.com/facebookresearch/votenet>.
34. Wikipedia Contributors. 2020. Training, validation, and test sets. *En.wikipedia.org*.  
[Online]. [Accessed 29 August 2020]. Available from:  
[https://en.wikipedia.org/wiki/Training,\\_test,\\_and\\_validation\\_sets](https://en.wikipedia.org/wiki/Training,_test,_and_validation_sets).
35. Brownlee, J. 2020. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. *Machine Learning Mastery*. [Online]. [Accessed 29 August 2020]. Available from: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=Adam%20is%20an%20optimization%20algorithm,iterative%20base d%20in%20training%20data.&text=The%20algorithm%20is%20called%20Adam>.
36. Rosebrock, A. 2020. Intersection over Union (IoU) for object detection - PyImageSearch. *PyImageSearch*. [Online]. [Accessed 29 August 2020]. Available from: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>.
37. Hui, J. 2020. mAP (mean Average Precision) for Object Detection. *Medium*. [Online]. [Accessed 29 August 2020]. Available from:  
[https://medium.com/@jonathan\\_hui/map-mean-average-precision-for-object-detection-45c121a31173](https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173).
38. Schnabel, R., Wahl, R. and Klein, R. (2007), Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26: 214-226. doi:[10.1111/j.1467-8659.2007.01016.x](https://doi.org/10.1111/j.1467-8659.2007.01016.x)
39. S. Song and J. Xiao, "Deep Sliding Shapes for Amodal 3D Object Detection in RGB-D Images," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 808-816, doi: 10.1109/CVPR.2016.94.



40. X. Chen, H. Ma, J. Wan, B. Li and T. Xia, "Multi-view 3D Object Detection Network for Autonomous Driving," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 6526-6534, doi: 10.1109/CVPR.2017.691.
41. Ku, J., Mozifian, M., Lee, J., Harakeh, A. and Waslander, S. 2020. Joint 3D Proposal Generation and Object Detection from View Aggregation. *Arxiv.org*. [Online]. [Accessed 29 August 2020]. Available from: <https://arxiv.org/pdf/1712.02294.pdf>.
42. Wikipedia Contributors. 2020. Convolutional neural network. *En.wikipedia.org*. [Online]. [Accessed 29 August 2020]. Available from: [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network).
43. Willow Garage. 2020. Object Recognition Using Linemod — object\_recognition\_tutorials. *Wg-perception.github.io*. [Online]. [Accessed 29 August 2020]. Available from: [https://wg-perception.github.io/ork\\_tutorials/tutorial03/tutorial.html](https://wg-perception.github.io/ork_tutorials/tutorial03/tutorial.html).
44. Rosebrock, A. 2020. (Faster) Non-Maximum Suppression in Python - PyImageSearch. *PyImageSearch*. [Online]. [Accessed 29 August 2020]. Available from: <https://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/>.
45. Zhang, Z., Yang, H., Sun, B. and huang, Q. 2020. H3DNet: 3D Object Detection Using Hybrid Geometric Primitives. GitHub. [Online]. [Accessed 30 August 2020]. Available from: <https://github.com/zaiweizhang/H3DNet>.
46. Anon 2020. Asus Xtion Pro Live ROS Installation. RoboR@m. [Online]. [Accessed 30 August 2020]. Available from: <https://roboram.wordpress.com/2016/04/19/asus-xtion-pro-live-ros-installation/>.

## **Appendix A**

### **External Materials**

#### **A.1 External Materials**

Qi et al's github repository for their VoteNet Architecture

- <https://github.com/facebookresearch/votenet>

Tutorial for object detection using ROS and Linemod

- [https://wg-perception.github.io/ork\\_tutorials/tutorial03/tutorial.html](https://wg-perception.github.io/ork_tutorials/tutorial03/tutorial.html)

Github repository for pre-processed SUN RGB-D dataset

- <https://github.com/zaiweizhang/H3DNet>
- [https://drive.google.com/file/d/1P\\_uFQcvVFf10TLxjlaFMfjto8ZHON-N2/view?usp=sharing](https://drive.google.com/file/d/1P_uFQcvVFf10TLxjlaFMfjto8ZHON-N2/view?usp=sharing)

Numpy: library for arrays

Torchvision library: build CNN models and architectures

#### **A.2 Code for the project**

**Source code is on github and the data are on google drive**

- <https://github.com/sc19aas/3D-object-detection>
- [https://drive.google.com/drive/folders/1ScYig5Jx61cnWL7RnpE5L3qQyNgA\\_OiR?usp=sharing](https://drive.google.com/drive/folders/1ScYig5Jx61cnWL7RnpE5L3qQyNgA_OiR?usp=sharing)

## **Appendix B**

### **Ethical Issues Addressed**

The major ethical issue pertaining to this project is Surveillance Ethics. Surveillance involves engaging in prolonged entity spying for any reason, Entity could include person or group [5]. One of the most popular uses of object detection is for facial recognition [6] where features from the face are extracted and classified.

With regards to ethics involving this technology, development bias and ethics surrounding the use are two of the biggest concerns [7]. Facial recognition relies heavily on datasets which is used to train deep neural networks. Getting datasets that have diverse target objects to help make the neural network diverse and robust is difficult, few and far between [7]. It is usually skewed towards one demographic which leads to accuracy issues in detection [7]. The need to curb the bias led to some ethical questions about privacy, ownership and how security agents would use facial data [7].

In this case for this project, it would involve using or collecting videos or images of people without their consent, targeting a group of people for discrimination.

At the moment I do not plan on using any personal data. My data doesn't consist of human images or any sensitive information. Therefore I do not expect any ethical issues since I'm using datasets consisting of objects that are publicly available.

With regards social issues, object detection is a driving force in autonomous vehicles, with little margin for error, there are discussions on how safe this technology is for driving [8]. People argue that they reduce accidents being that most accidents are due to human carelessness according to Google about their driverless cars [9]. Those who argue for autonomous cars, emphasize its efficiency and safety while those against, are sceptical about the safety issues.