People Detection from LiDAR Point Clouds collected by Mobile Robots in Cluttered Indoor Environments

by

Hshmat Sahak

Supervisor: Tim Barfoot
April 2024

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

People Detection from LiDAR Point Clouds collected by Mobile Robots in Cluttered Indoor Environments

by

Hshmat Sahak

Supervisor: Tim Barfoot

April 12, 2024

Abstract: People detection in indoor environments is pivotal for mobile robots as it ensures safe and efficient navigation in dynamic environments. From autonomous delivery drones navigating crowded urban areas to robotic assistants aiding in hospitals, accurate people detection enables robots to interact seamlessly with humans, enhancing productivity and safety across various domains.
Existing work has shown that a SLAM classification pipeline can be used to annotate points in a scene as ground, permanent, short-term movable or long-term movable. A KPConv network can then be trained using the annotated data to detect dynamic (short-term movable) points with high accuracy. This work seeks to study heuristic and neural approaches to people detection using the outputs of these methods. Specifically, we investigate whether it is better to have the heuristic detector as an additional module on top of the network predictions, or use the heuristic on the SLAM classification output to develop a ground truth (point cloud, boxes) dataset to be used for downstream training.
The method we pursue is as follows. We focus on the UTIn3D_A + UTIn3D_B dataset. We first filter annotated frames by the point classification, selecting only dynamic points. Then we apply the heuristic algorithm to find the people. The gives a dataset of point cloud points and boxes that we use to train a SECOND network. We call this method 1. Next, we train a KPConv network to predict point labels from SLAM outputs. During validation, we simply add heuristic classification on top of the points predicted as people. We call this method 2. Note that for both methods, we are able to treat dynamic points as people points because people are the only dynamic actors in the UTIn3D scenes.
At the current iteration of the study, we conclude that method 2 - applying the heuristic as an additional module on top of KPConv predictions, outperforms training an end-to-end neural network. However, we provide potential improvements to method 1 that may narrow the gap between the two methods.

Acknowledgements: I would like to thank Professor Tim Barfoot, who granted me the amazing opportunity to do my undergraduate thesis project under his supervision. I would also like to thank Binbin Xu, Hugues Thomas and Daniil Lisus for answering my questions about existing work.

I am indebted to my family for their unconditional love and support. My parents deserve special recognition for their unwavering support throughout my undergraduate endeavor.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

Autonomous mobile robots are becoming increasingly popular in indoor environments in the real-world, for example, in hospitals [1] and warehouses [2]. People detection while traveling is an important part of the robot's navigation stack. Specifically, human detection is important for motion forecasting and therefore crucial for path planning to ensure the robot avoids collisions or blocking a person's expected trajectory. For effective perception in indoor environments, 3D LiDAR is often used for sensing the environment, as it gives a high-resolution point cloud with great speed and precision, generating a descriptive geometry of the environment [3]–[5]. However, the irregular structure of the data and large search space makes simultaneously detecting people and mapping the environment difficult. This is especially true when a person is interacting with another dynamic (e.g., another person) or static (e.g., chair) object [6]. Figure 1 depicts the overall goal in robot navigation with respect to obstacle avoidance. Figure 2 shows the kind of data we work with in this project.

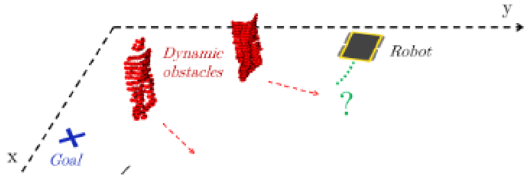


Figure 1: People Detection in Indoor Environments is crucial for path planning.

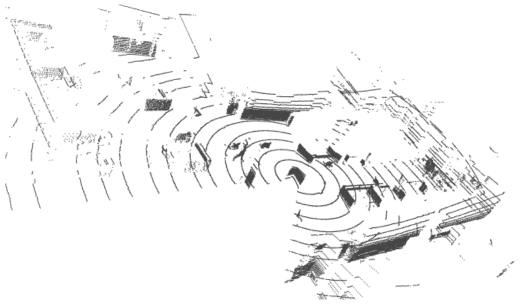


Figure 2: Compared to images, the usual input to CNNs, LiDAR is difficult to deal with due to its sparsity and irregularity.

1.2 Research Gap

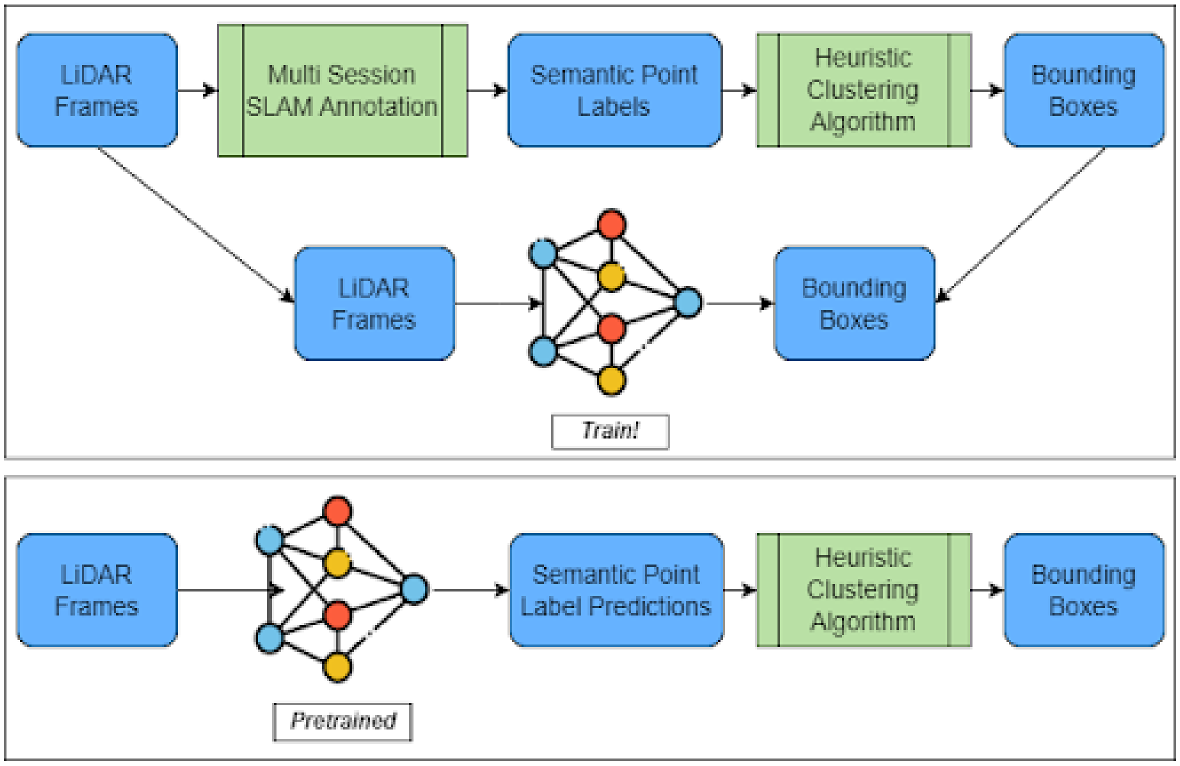
There are several **classical/heuristic** and **neural** methods for human detection from 3D LiDAR data. *Classical* methods depend on cluster detection followed by cluster classification. The literature on cluster detection is vast, and several implementations are included in standard Python libraries. Cluster classification methods, like the Support Vector Machine (SVM), depend on hand-crafted features, but have low computational complexity so detection can occur in real time. Features useful for cluster classification include geometric and motion features [7], human slice features and distribution of reflections [8], and 2D histograms from projection of 3D lidar [9]. In contrast, *neural* methods require more compute but are often more accurate. The best-performing techniques convert point clouds to voxel grids, combines adjacent voxels to form cuboids and pass cuboids as input to a Convolutional Neural Network (CNN) to label all points in the center voxel of the cuboid [10].

Although neural methods outperform heuristic methods on point classification, identifying a bounding box to mark the pose of each person given these point classifications can sensibly be done with heuristic clustering methods *or* with self-supervised learning. Indeed, there exist pipelines that are able to classify points in a point cloud as static (do not move), short-term movable (dynamic, like people), and long-term movable (movable objects, like chairs) [11]. We expect this to make the task of people detection easier. To our knowledge, there is no previous work that compares heuristic and neural methods on people detection from LiDAR point clouds once points are already labelled as ground, static, movable or dynamic.

1.3 Goal of Thesis Work

The goal of this work is to compare heuristic and neural methods on people detection from 3D LiDAR point clouds given points are already labelled as static, short term movable or long term movable. Our main objective is to compare the two methods on performance. We envision a navigation stack whereby the robot moves in a people-chairs-tables environment by detecting its environment using lidar, identifying dynamic points, and outputting a bounding box for each person. The robot can then use standard path planning using these bounding boxes (and potentially, their projected future locations) as obstacles to avoid in order to safely move around. For sake of simplicity, we do not distinguish between two people who are together in a scene as being identified by two separate bounding boxes or one bigger box. We argue that planning algorithms that treat the two people as one big bounding box approximate the motion of the group as they move together anyways.

The purpose of this document is to record the progress in the thesis research. Readers should be able to understand the goal of the project, related background material, methods, findings and future work.



LiDAR Frames
Multi Session SLAM Annotation
Semantic Point Labels
Heuristic Clustering Algorithm
Bounding Boxes
LiDAR Frames
Bounding Boxes
Train!
LiDAR Frames
Semantic Point Label Predictions
Heuristic Clustering Algorithm
Bounding Boxes
Pretrained

Figure 3: Self-supervised learning pipeline.

2 Background

Although existing literature does not compare heuristic and neural methods while also incorporating dynamic point classification as part of the pipeline, there are numerous studies [10], [12]–[19] that individually investigate either heuristic or neural methods for people detection, which *do not* incorporate point classification. Now, the point classification module can be viewed as simply returning a subset of lidar points (corresponding to dynamic objects) for the classification task. Thus, this module does not restrict our choice of heuristics or networks for detection; so, existing papers are still directly relevant to our work.

Figure 3 shows our self-supervised pipeline. We first use an off-the-shelf point classification network [11] to detect all short-term movable points. Since our dataset is collected in a simple people-chair-table environment, these points should all correspond to people. On top of this, we add a heuristic clustering module to then detect the bounding boxes of people from the output of the first module (the "people points"). The heuristic strategy (method 2) would then be to simply run these 2 modules for any LiDAR frame during inference. The neural approach (method 1) would be to run the bounding box detector on real navigation data- specifically, on points labelled as dynamic by the SLAM classification algorithm. The resulting input and output is then used as a dataset to train a model to predict bounding boxes directly from the point cloud or point cloud sequences. It is referred to as "self-supervised" because the training labels come from the data itself.

In the literature review that follows, we will explain in detail the 3 previously mentioned components - the dynamic point classification module, heuristic people detection from people points, and neural network training for bounding box detection. Most of the work we present performs well for data collected from both indoor and outdoor environments.

2.1 Dynamic Point Classification from LiDAR Point Cloud

Our implementation of lidar segmentation into ground, permanent, short-term movable and long-term movable points comes directly from [11]. In that paper, the authors propose a self-supervised learning method for the segmentation of lidar point clouds in a multi-session setup. They are able to label each point as one of the 4 previously mentioned classes; where the short term movables should be people and long-term movables should be chairs and tables. We use their automated annotation process, which labels 3D point clouds from previous sessions, to obtain the desired semantic information. Lidar frames of successive sessions are annotated using 2 main modules, PointMap and PointRay, discussed below.

PointMap: The purpose of PointMap is to obtain a point cloud map that can be used for the annotation process. The algorithm involves a two-stage procedure where we first use Iterative Closest Point (ICP) to align a frame to the map, and then a mapping function that updates the map with the aligned frame. The ICP algorithm uses the same elements as [20], specifically the data filters, matcher, outlier rejection, distance function and convergence tests. The choices for each element are inspired by previous work and their specific lidar sensor.

The second component involves updating the map with an aligned frame. The map is defined by a sparse 3D grid, with a tune-able cell size. To regulate the density of points, only one point is kept per cell, with its normal and a score $s \in [0, 2]$. Updating the map with a frame follows certain rules. First, once a cell is assigned a point, it will not change (this is to reduce drift). The cell associated with a certain position x is:

$$vox(x) = \left\lfloor \frac{x - x_{origin}}{dl_{map}} \right\rfloor \in \mathbb{Z}_3$$

However, normals are updated by keeping the normal with the highest score. The normal is computed using radius neighbours in spherical coordinates, keeping 3 consecutive lines of a scan. The score is constructed to capture two ideas - i) normals should be computed from a close distance R, but ii) only if the lidar position is facing the surface. This is captured by the following score function:

$$s = \begin{cases} \frac{\left(\frac{\pi}{2} - \theta\right)}{\left(\frac{\pi}{2} - \theta_0\right)}, & \text{if } \theta > \theta_0 \\ \frac{1}{1 + e^{-(R - R_0)^2}}, & \text{else} \end{cases}$$

where $\theta_0 = \frac{5\pi}{12}$ is the minimum incidence angle for the sensor to be considered facing a surface and $R_0 = 2m$ is the ideal distance for normal computations if the sensor is facing the surface.

PointRay: In this stage, they ray-trace movable probabilities. Movable probabilities are the opposite of occupancy probabilities, so it is the likelihood of a voxel corresponding to free space. Movable probabilities are obtained by assessing distance between points in each lidar frame and the map of the environment. Specifically, the technique in [21] is used; whereby they project both the frame and map in the frame spherical coordinates and consider the distance between frame and map points.

Specifically, the frame and map are both projected to a frustum grid to encode free space. The frustum grid is a 2D grid in the θ and ϕ spherical coordinates, and serves as a lidar depth image whose pixels store the smallest point distances to the lidar origin. Then, the radius ρ of each point in the cropped map is compared to the depth saved in its corresponding frustum, ρ_0. To be updated as free space, the point must meet two conditions:

$$cond_A : \rho < \rho_0 - \text{margin}(\rho_0)$$
$$cond_B : |n_z| > cos(\beta_{\min}) \quad \text{OR} \quad \alpha < \alpha_{\max}$$

where $\text{margin}(\rho_0) = \rho_0 \cdot max(d\theta, d\phi)/2$ is a tolerance buffer on the ρ parameter, n_z is the vertical component of the point normal, and α is the angle of incidence between the lidar ray and this normal. The first condition ensures that a planar surface with incidence angle less than $\pi/4$ will not be updated as free space. The second captures the fact that we do not want to update for extreme incidence angles.

After repeating the above process for a full navigation session, we can get the final movable probabilities for every point x_i of the map as follows:

$$p_{mov}(x_i) = \begin{cases} \sum_{k<n_i} p_i^k / n_i, & \text{if } n_i > n_{min} \\ 0.5, & \text{else} \end{cases}$$

where n_{min} is the minimum number of times we must see a point to assign it a movable probability.

Multi-Session Annotation: PointMap and PointRay can be used together to annotate multiple sessions. To achieve this, we follow the steps below, which are visualized in Figure 4.

1. To achieve alignment of multiple sessions (required by PointRay), we have them localize against the same initial map. In the setting described in [11], it was reasonably assumed that the environment was mapped at the beginning of the project. This is done by an initial mapping tour covering the entire environment.

2. We obtain a refined map from the initial map by applying PointRay to it and removing points with higher movable probability than 0.9. The points from the refined map are then labelled as permanent or ground based on the ground extraction algorithm RANSAC.

3. PointMap is used to create an aggregated map of all frame points. This is the buffer map. The role of the buffer is to be annotated and then have labels transferred back to individual frames by nearest neighbour interpolation. To do this, PointRay is first used to get the movable probabilities of buffer points. Then, for points seen more than n_{min} times, these probabilities are classified as either *shortT* or *longT*, depending on some threshold. The points seen less than n_{min} times are labelled as *uncertain*.

4. The final point labels are projected back to individual frames.

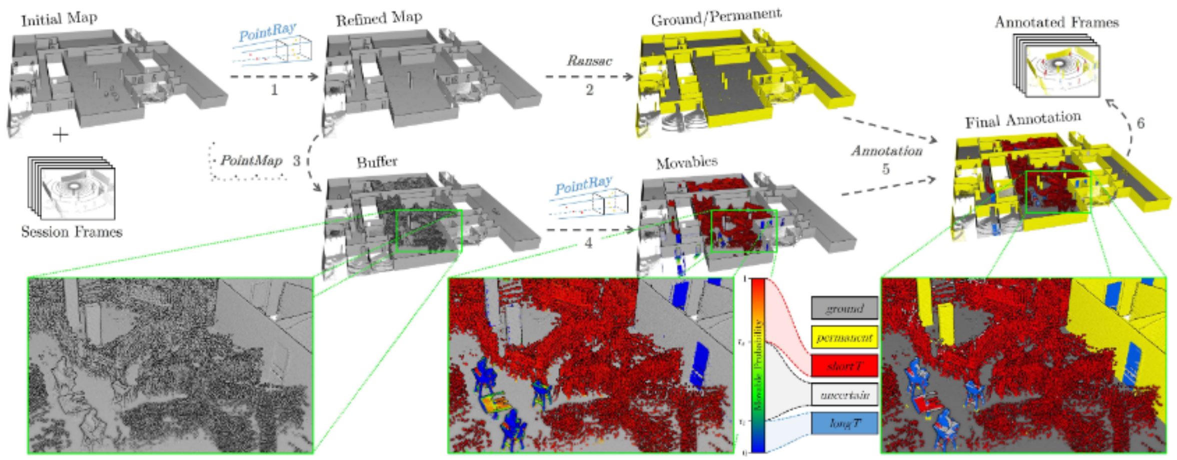


Figure 4: Multi-session annotation procedure for frames of one session

2.2 Heuristic Methods for Cluster Detection in LiDAR Point Clouds

Several papers [12], [13], [22] have been published that highlight the efficacy of non-neural methodologies for human detection. These approaches possess an inherent appeal due to their simplicity in implementation and their ability to address the problem from a comprehensible, human-oriented perspective. This stands in contrast to neural networks, which are often perceived as black boxes during the inference process. Furthermore, heuristic techniques provide an alternative to labelling point clouds, which is both tedious and prone to error. Offline annotation is particularly infeasible for complex real-world scenarios, as we may have to train the same system under different operational environments. In this section, we will go over the main techniques presented for cluster detection, the most popular heuristic approach [22].

2.2.1 Most Popular Clustering Algorithms

It is common for clustering to be done in two phases. The first phase is ground point identification. The second is clustering of non-ground points. The second phase can be tackled by well-known clustering algorithms that are easy to implement, simple and have high execution speeds. 3 examples of these are the euclidean cluster extraction, DBSCAN, and Mean-Shift.

1. Euclidean Cluster Extraction [23]

To achieve cluster detection from 3D data in situations where clusters can be of different sizes, we need to define what an object point cluster is, and how to differentiate it from other clusters. This is done mathematically; specifically, we say a cluster $O_i = \{p_i \in P\}$ is a distinct cluster from $O_j = \{p_j \in P\}$ if $\min \|p_i - p_j\| \geq d_{th}$, where d_{th} is an imposed maximum distance. Intuitively, if the minimum distance between a set of points p_i and a set of points p_j is larger than some threshold, then the points in each set belong to different clusters. The algorithmic steps to achieve this are as follows:

1. create a kd-tree representation for point cloud dataset P
2. initialize empty list of clusters C and a queue of points to be checked, Q
3. for every point $p_i \in P$, do the following
   - add p_i to the current queue Q;
   - search for the set of neighbours of p_i in a sphere with radius $r = d_{th}$;
   - for each neighbour, check if it has already been processed; if not, add to Q
   - when all points in Q have been processed, Q is added to cluster list C, and Q is reset to an empty list
4. algorithm terminates when all points $p_i \in P$ have been processed and are now part of the list of point clusters C.

An example output of such a clustering algorithm is shown in Figure 5.

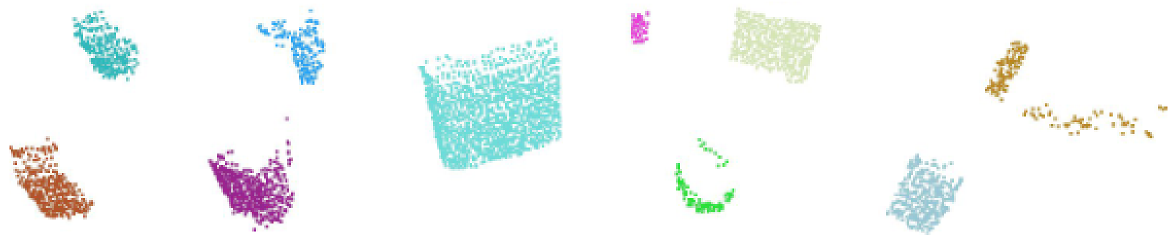


Figure 5: Sample Euclidean clustering results for non-ground points

2. DBSCAN [24]

In their paper, the authors of [24] present a clustering algorithm, DBSCAN, which relies on a density-based notion of clusters that is designed to discover clusters of arbitrary shapes. The success of the DBSCAN algorithm relies on distinguishing core from boundary points, and connecting border points to corresponding core points that contain these border points. The intuitive notion of ”core points”, ”border points” and clusters are captured through the following definitions:

Definition 1: (Eps-neighbourhood of a point) The Eps-neighbourhood of point p, denoted by $N_{eps}(p)$, is defined by $N_{eps}(p) = \{q \in D \mid dist(p, q) \leq \text{Eps}\}$

Definition 2: (directly density-reachable) A point p is directly density-reachable from a point q given parameters Eps, MinPts if $p \in N_{eps}(q)$ and $\|N_{eps}(q)\| \geq MinPts$. For two core points, this is symmetric. For a core and border point, it is not. This is shown in Figure 6 below.

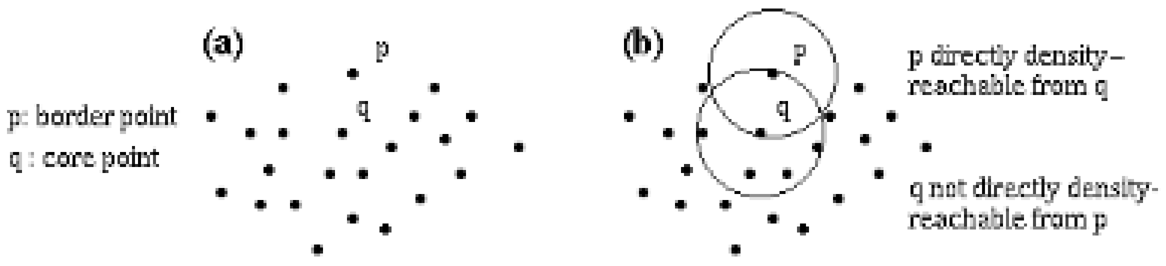

(a)
p
q
p: border point
q : core point
(b)
p
q
p directly density-reachable from q
q not directly density-reachable from p

Figure 6: Core and Border points in DBSCAN are treated separately

Definition 3: (density-reachable) A point p is density-reachable from q given Eps and MinPts if there is a chain of points $p_1, p_2, ..., p_n, p_1 = q, p_n = p$ such that p_{i+1} is directly density reachable from p_i.

The relation above is transitive, but not symmetric in general. Obviously, it is symmetric for two core points, but Figure 7 gives an example of the asymmetric case where we can have a border and core point.

Although two border points of the same cluster should be identified as belonging to one cluster, the above definition does not make them density reachable. The authors therefore present the notion of density-connectivity, to cover this relation, by capturing the fact that they must be density-reachable from some core point.

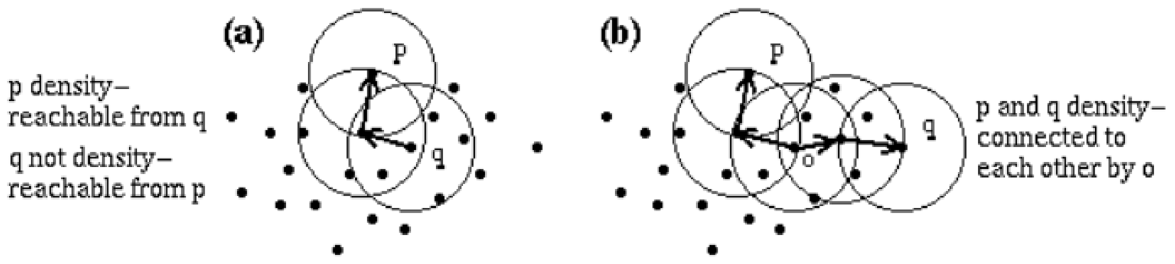

(a)
p
q
p density-reachable from q
q not density-reachable from p
(b)
p
o
q
p and q density-connected to each other by o

Figure 7: Density-reachability and density-connectivity

Definition 4: (density-connected) Two points p and q are density-connected if there is a point o such that p and q are both density-reachable from o given Eps and MinPts. This definition is transitive and symmetric.

Now, we are equipped to define clusters from a density-based perspective. Specifically, a cluster is a set of density-connected points which is maximal in the sense of density-reachability. In the definitions and pseudocode that follow, Noise is defined as the set of points in D not belonging to any clusters.

Definition 5: (Cluster) Let D be a database of points. A cluster C is a set of points satisfying the following:

1. Maximality: $\forall p, q$; if $p \in C$ and q density-reachable from p, then $q \in C$
2. Connectivity: $\forall p, q \in C$, p is density-connected to q

Definition 6: (noise) Noise is the set of points that don't belong to any cluster.

The definitions 1-6 motivate the following basic version of the DBSCAN algorithm:

SetOfPoints is the set of all points in the dataset. Eps and MinPts are global density parameters; they can be optimized manually or using heuristics outlined in [24]. The most important function, ExpandCluster, is presented below:

Calling $SetOfPoints.regionQuery(Point, Eps)$ returns the eps-neighbourhood of $Point$ in $SetOfPoints$. This is supported by methods such as $R^* - trees$. The runtime of one query is $O(logn)$ and there are n points, so the runtime of DBSCAN is $O(nlogn)$.

3. Mean-Shift [25]

```
DBSCAN (SetOfPoints, Eps, MinPts)

// SetOfPoints is UNCLASSIFIED
  ClusterId := nextId(NOISE);
  FOR i FROM 1 TO SetOfPoints.size DO
    Point := SetOfPoints.get(i);
    IF Point.ClId = UNCLASSIFIED THEN
      IF ExpandCluster(SetOfPoints, Point,
             ClusterId, Eps, MinPts) THEN
        ClusterId := nextId(ClusterId)
      END IF
    END IF
  END FOR
END; // DBSCAN
```

Figure 8: DBSCAN Algorithm

```
ExpandCluster(SetOfPoints, Point, ClId, Eps,
               MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeClId(Point,NOISE);
    RETURN False;
  ELSE   // all points in seeds are density-
         // reachable from Point
    SetOfPoints.changeClIds(seeds,ClId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                        Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.ClId
              IN {UNCLASSIFIED, NOISE} THEN
            IF resultP.ClId = UNCLASSIFIED THEN
              seeds.append(resultP);
            END IF;
            SetOfPoints.changeClId(resultP,ClId);
          END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

Figure 9: Expand Cluster method

The mean shift algorithm assigns points to clusters by iteratively pushing points towards certain "modes", or areas where the underlying probability distribution has gradient zero, which correspond to the most dense points in the distribution.

Unlike K-means algorithm, Mean Shift does not require knowledge of the number of clusters. This is determined by the number of data points and how they are arranged. Thus, mean shift is a non-parametric density-based algorithm, particularly useful when the clusters have arbitrary shapes and not well separated by linear boundaries.

The basic idea behind mean shift is that points are iteratively shifted to the mean of the points within a certain neighbourhood of that point. Eventually, the points will converge to a set of local maxima, which identify the clusters of our data. The exact steps are outlined below; and the complex mathematical derivation and proof of convergence can be found in [25].

1. Initialize data points as cluster centroids
2. For each data point, calculate mean of all points within a certain radius, and shift this data point to the mean.
3. Cluster centroids are points that do not move after convergence
4. Assign data points to clusters they map to

An example of running the mean shift algorithm is shown in Figure 10.

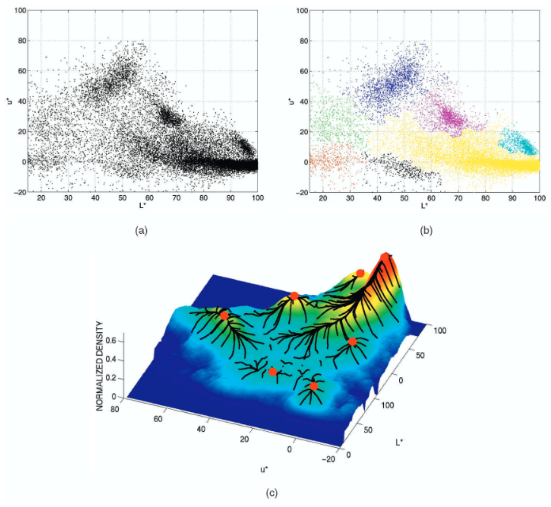


Figure 10: Sample Result of Mean Shift Algorithm

2.2.2 Other Well-Known Clustering Algorithms

Other major clustering algorithms include scan line run [26] and depth information [27]. In scan line run, the segmentation occurs in two steps. In the first step, it extracts the ground surface. In the next, it clusters non-ground points using a two-run connected component labelling technique from binary images. In depth clustering, 3D lidar scans are first converted to 2D range images, then segmentation is done on the 2D range image. The method we propose next, proposed in the paper "Online learning for 3D LiDAR-based human detection: experimental analysis of point cloud clustering and classification methods" [22] outperforms these algorithms and euclidean-based clustering techniques.

2.2.3 Novel Point-cloud Cluster Detector

Here, we will go over the point cloud cluster detector proposed by [22]. The input of their clustering algorithm is a 3D lidar point cloud $P = p_i \mid p_i = (x_i, y_i, z_i) \in R^3, i = 1, ...I$, where I is the total number of points in a single scan. First, ground points are discarded by removing points with $z < z_{min}$. Obviously, this only works well if the ground is flat, but the algorithm is not meant to care much about the ground point detections, as this can be improved as a separate module.

The main clustering is done similar to [23]. Non-overlapping clusters of adjacent points are identified based on their euclidean distance. Specifically, the following condition must hold for clusters C_i and C_j to be distinct:

$$C_i \cap C_j = \emptyset, \text{for } i \neq j => min\|p_i - p_j\|^2 \geq d^*$$

$$\text{for } p_i \in C_i, p_j \in C_j, \text{ and } d^* \text{ a distance threshold.}$$

Now, this is fast and simple, but can be inaccurate when the density of points varies with the distance from the sensor. In lidar applications, we expect points are less dense the further away from the sensor. With small threshold d^*, single objects could be split into multiple clusters. With large threshold, multiple objects could be grouped together as one cluster. To deal with sparser point clouds the further we move from the sensor, the authors use a threshold that is dependent on the scan range r:

$$d^* = 2 \cdot r \cdot tan\frac{pi}{2}$$

To reduce computational complexity of clustering, the lidar scan range is divided into nested circular regions around the sensor, and a constant distance threshold is assigned to each of them. The corresponding radius of the regions are identified by taking the inverse of the above equation.

Finally, clusters that do not make sense as humans are filtered out. The authors use the filter $\bar{C} = \{C_j \mid 0.2 \leq w_j \leq 1, 0.2 \leq d_j \leq 1, 0.2 \leq h_j \leq 2\}$, which ensures the volume containing C_j is not too small or too large. The clustering algorithm is implemented using based on a k-d tree, the time complexity is O(log n) per point, or O(nlogn) overall.

2.3 Human Classification from LiDAR Point Clusters

In human detection pipelines, it is common to perform human classification after cluster detection; that is, predict whether the cluster is a person or not. Since our project setup ensures that the only clusters of dynamic points are people, we do not need to implement a classification layer. However, we provide a brief overview of common approaches for sake of completion, and for possible extensions to cases where our environment has more than one type of dynamic actor.

A common method is to train a classifier offline, and then apply the classifier to sensor data during navigation. [7] introduced 7 features indicative of humans in clusters and used them to train a SVM. [8] showed significant improvements by adding two more features considering the 3D human shape and clothing material (using reflected laser beam intensities). [28] supported previous works by using 8 of the features for human detection in unstructured environments, excluding the intensity feature for hardware support reasons. [29] improved the quality of geometric features, showing the sensitivity of a classifier to human shape features. [30] used a sliding window technique. They divided the 3D space into sparse feature grids and trained a SVM on grid features relating to the occupancy of the cells, distribution of points in them and intensity values of the points. There also exist methods that project 3D lidar to 2D depth images [31], [32], and then either use standard supervised learning or unsupervised feature learning.

The work done by [22] launches experiments to investigate which features is best to use for a SVM model. This is useful as SVMs have a solid theoretical foundation and are good with small sample datasets, which we suspect is common in robotics. The features they found most useful are shown in the Table 1. Ultimately, they found that using features (f1, ... f4, f8, ... f10) gave the best balance between performance and efficiency.

Feature	**Description**
f1	Number of points included in the cluster
f2	Minimum cluster distance from the sensor
f3	3D covariance matrix of the cluster
f4	Normalized moment of inertia tensor
f5	2D covariance matrix in 3 zones including the upper half, the left and right lower halves
f6	The normalized 2D histogram for the main PC plane
f7	The normalized 2D histogram for the secondary PC plane
f8	Slice feature for the cluster
f9	Reflection intensity's distribution (mean, standard dev. and normalized 1D histogram)
f10	Distance from the centroid of each slice to the sensor

Table 1: Features relevant for training a SVM for cluster classification. Features in italics give best results while also allowing real time classification.

The inconvenience with offline training methods is that the pretrained classifier is not as effective when the robot moves to a different environment. This is problematic as robotic tasks often rely heavily on the specific environment, for example, indoor cluttered environments in our case. Some work has already proposed methods using less manual annotation. For example, [33] presents a semi-supervised framework which only requires a small set of object tracked over time. Other works present classifier-free methods; for example, [34] introduces a surface matching technique for people detection and a Extended Kalman filter to assist with detection in future frames. This method is particularly suited for the simple case of moving humans. While we could use one of these semi-supervised or classifier-free approaches, it is much simpler to assume that our dynamic point classifier identifies the dynamic points (people) accurately, and thus there is no need for any manual annotation as we will only cluster people anyways.

2.4 Human Detection from Point Clouds using Deep Learning

Section 2.2 served to discuss heuristic clustering methods. As stated previously, neural networks can be used for human clustering or pose estimation from 3D lidar point clouds. In this section, we will first go over possible data representations that are suited as inputs to ML models. Then, we will present a chronological ordering of different detection techniques. Lastly, we will go over PointNet and VoxelNet, the main architectures from which people detection from single lidar frames is based off.

2.4.1 Data representations for 3D LiDAR Object Detection

The raw point cloud from a single frame must be formatted to an appropriate input for a ML model. There are 4 widely used data representations: point-based, grid-based, combination of point- and voxel-based, and range-based. A brief overview of each is given below.

Point-based 3D object detection: The general point-based object detection method consists of blocks for point cloud sampling and feature learning, followed by a prediction head. The sampling and feature learning blocks serve as the backbone while the prediction module directly estimates 3D bounding boxes from the sampled points. The general structure is shown in Figure 11

Grid-based 3D object detection: Grid-based approaches convert the point clouds into 3 grid representations - voxels, pillars and birds eye-view (BEV) feature maps. Feature extraction is done by applying 2D CNNs or 3D sparse neural networks on such grid representations. Finally, 3D objects are predicted from BEV grid cells. The general structure of grid-based object detectors are shown in Figure 12

Point-voxel based 3D object detection: In the single-stage point-voxel detection framework, point and voxel features are fused in the backbone network. In the two-stage framework, the 3D object proposals are obtained from the voxel-based detector, and then refined using points sampled from the point cloud. The general structure is shown in Figure 13

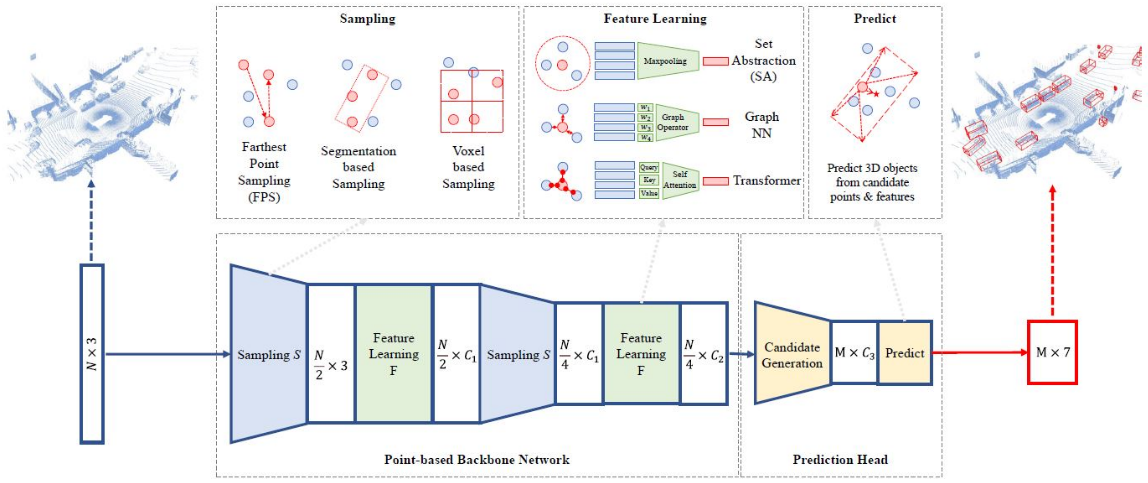


Figure 11: Structure of Point-based Detection Framework

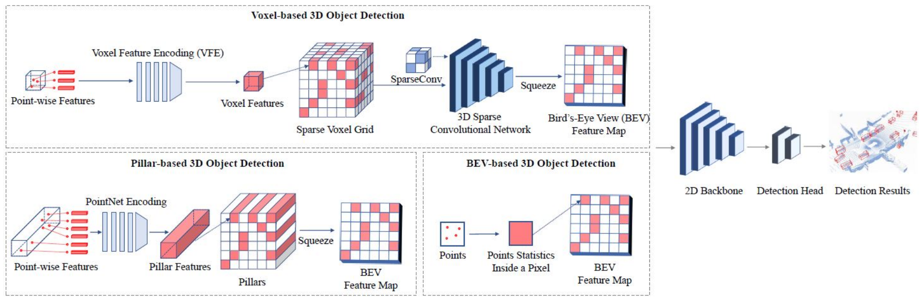

Voxel-based 3D Object Detection
Voxel Feature Encoding (VFE)
Point-wise Features
Voxel Features
Sparse Voxel Grid
SparseConv
3D Sparse Convolutional Network
Squeeze
Bird's-Eye View (BEV) Feature Map
Pillar-based 3D Object Detection
PointNet Encoding
Point-wise Features
Pillar Features
Pillars
Squeeze
BEV Feature Map
BEV-based 3D Object Detection
Points
Points Statistics Inside a Pixel
BEV Feature Map
2D Backbone
Detection Head
Detection Results

Figure 12: Structure of Grid-based Detection Framework

Range-based 3D object detection: One category of range-based approaches directly predicts 3D objects from pixels in range images using standard 2D convolutions. The second category transforms features from range view to bird's eye view, and then detects 3D objects from the transformed view. The general structure is shown in Figure 14. Since point and voxel approaches outperform range-based methods, we do not pursue this data representation in our work.

A chronological ordering of the best lidar-based object detection methods is shown in Figure 15. For sake of simplicity, we will consider only point and voxel-based methods, and present sample architectures in the next section. For point-based detection, we will present the PointNet architecture. For grid/voxel-based detection, we will go over the VoxelNet architecture. Although these are not state-of-the-art in their respective

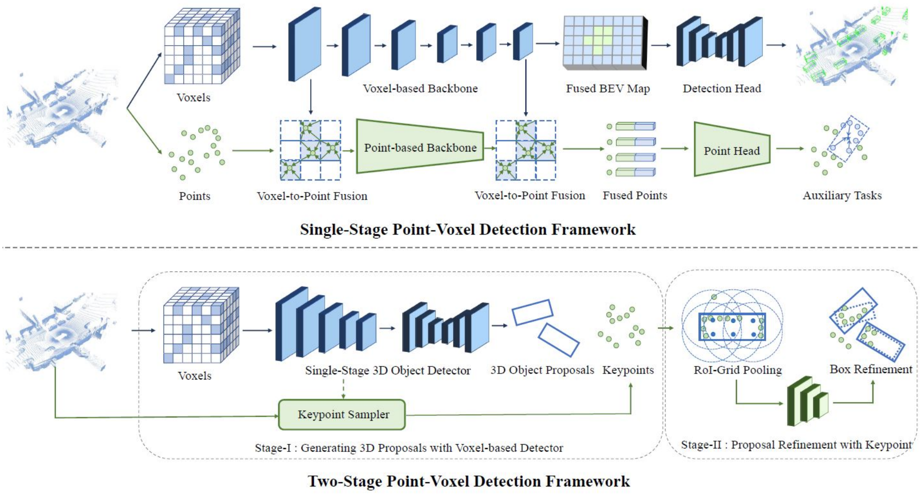


Figure 13: Structure of point- and voxel-based detection framework

detection methods, they serve as good examples of the fundamental idea behind point- and voxel-based approaches, respectively. Most advances in both point- and voxel-based approaches are variations of these works. A comprehensive list of sample works in each of these 4 domains is available in a survey paper for 3D object detection [35]. Although the survey paper focuses broadly on autonomous driving, pedestrian detection is certainly part of the autonomous driving framework and thus we expect the best-performing models to work well for our task.

2.4.2 Sample Point- and Grid/Voxel-based Architectures

PointNet [36]:

Networks that take the raw point cloud as input must strive to satisfy 3 properties:

1. *Unordered.* Unlike 2D images that have well-defined pixels or volumetric grids that can be voxelized, a point cloud is a set of points without a specific order. So, the network must strive to give the same output to any of N! permutations of the points. In other words, we want our model to be permutation invariant.

2. *Interaction among points.* The network should capture that the points are related to each other, usually through a distance function, which allows us to do clus-

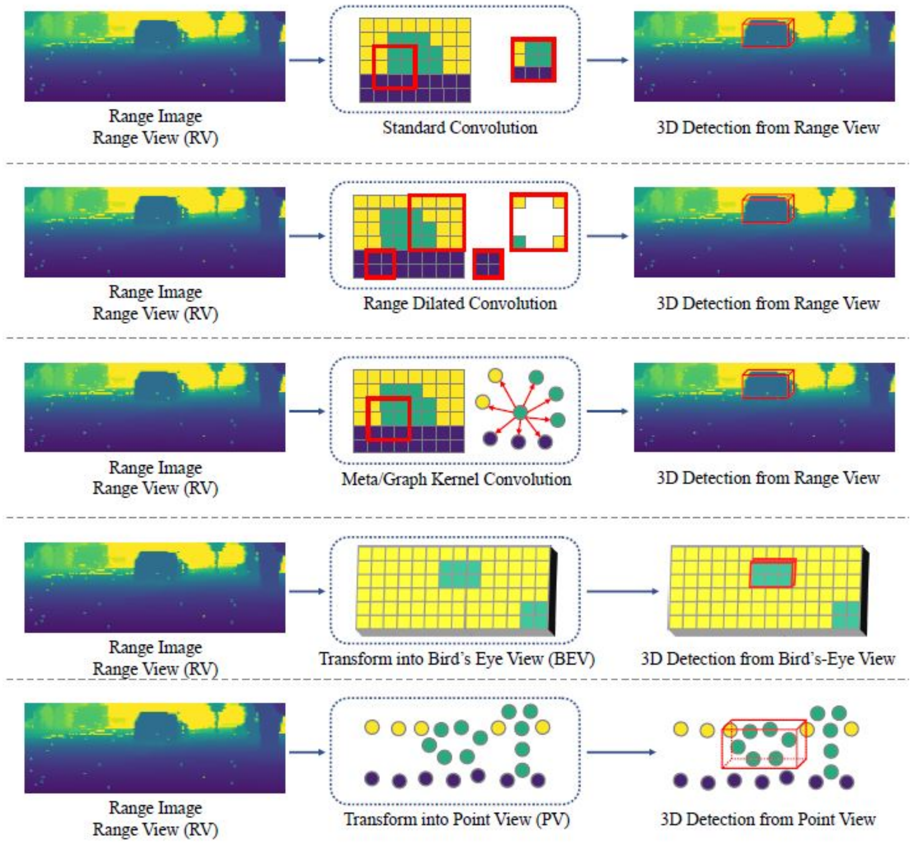

Range Image
Range View (RV)
Standard Convolution
3D Detection from Range View
Range Image
Range View (RV)
Range Dilated Convolution
3D Detection from Range View
Range Image
Range View (RV)
Meta/Graph Kernel Convolution
3D Detection from Range View
Range Image
Range View (RV)
Transform into Bird's Eye View (BEV)
3D Detection from Bird's-Eye View
Range Image
Range View (RV)
Transform into Point View (PV)
3D Detection from Point View

Figure 14: Structure of Range-based Detection Framework

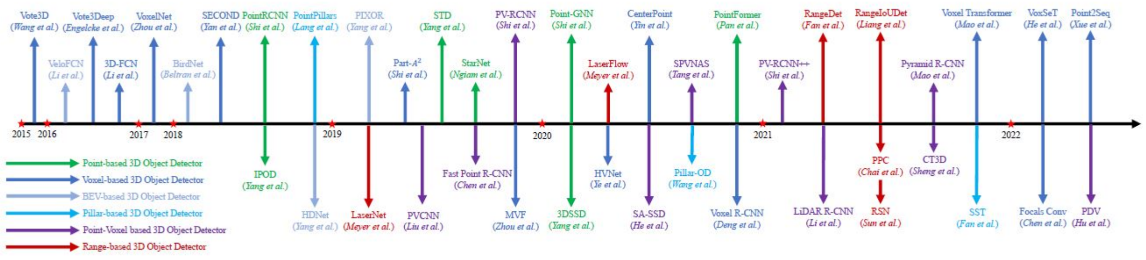


Figure 15: Chronological ordering of best-performing lidar object detection networks

tering and semantic segmentation. The model needs to capture local structures from nearby points, as well as the interaction between local structures.

3. *Transformation invariance.* The content of the data does not change with rotations and translations. The model should capture this by making sure the

classifications and segmentations are robust to such transformations.

The PointNet architecture seeks to achieve permutation invariance by using a global symmetric function, namely the MaxPooling function. To address the other constraints, highlights of the architecture include the MaxPool symmetry function to deal with unordered input, local and global information aggregation and a joint alignment network. These are shown in Figure 16

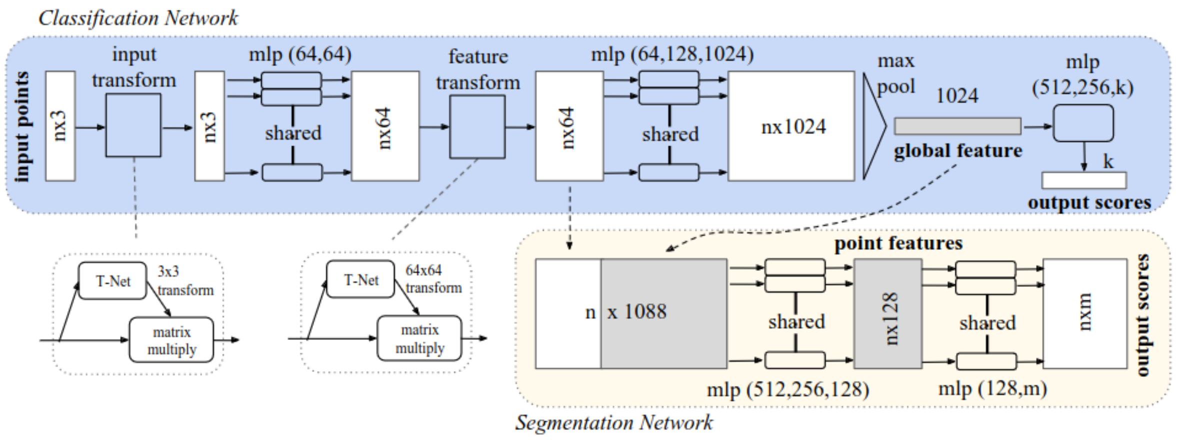


Figure 16: PointNet Architecture. The network takes n points, applies input and feature transformations to them, and aggregates point features by max pooling. Then it outputs probabilities of each of k classes using a multi-layer perceptron. The point and global features are also concatenated for semantic segmentation, where we predict one of m classes for each point.

VoxelNet [37]:

VoxelNet is a generic 3D detection framework that simultaneously learns a discriminative feature representation from point clouds and predicts accurate 3D bounding boxes, in an end-to-end fashion. It beats previous lidar-based 3D detection methods by a large margin, and is especially encouraging for finding pedestrians and cyclists from point clouds. The VoxelNet architecture is shown in Figure 17. The authors design a *Voxel Feature Encoding (VFE)* layer, which enables point-wise interactions within a voxel by combining point-specific features with a local aggregate. Stacking multiple VFE layers allows learning complex features for characterizing local 3D shape information. Specifically, VoxelNet divides the point cloud into equally sized voxels, encodes each voxel via the stacked VFE layers, and then performs 3D convolution layers on it which transforms the point cloud to a 4D volumetric tensor representation. Finally, a Region Proposal Network (RPN) [38] consumes the tensor and outputs the detection results. The RPN algorithm benefits from the sparse point structure and efficient parallel processing on the voxel grid.

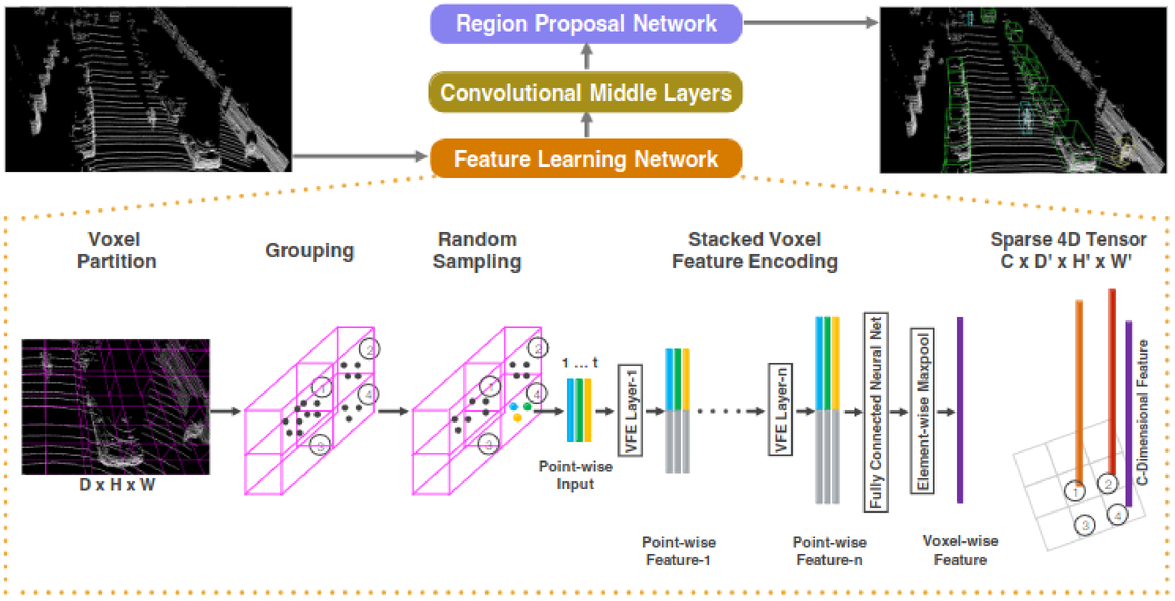


Figure 17: VoxelNet Architecture. The network takes in N points and converts the representation to a 4D tensor. The convolution middle layers process the tensor to capture spatial context, and a RPN generates the final detection.

2.5 Analysis of Literature: A Clear Path Moving Forward

Our literature review provides a clear path forward. The common and best-performing clustering algorithms have easily accessible implementations. State-of-the-art (SOTA) neural network object detectors are available as well. Voxel-based detectors generally outperform point-based detectors on pedestrian tracking [37]. Thus, a simple, promising path is to use an available pipeline for the heuristic clustering and finetune a publicly available VoxelNet-based architecture for our neural network approach. To determine whether the heuristic method is better or and end-to-end network, we will compare the output of each technique on a hold-out validation set. 2.1.

3 Methodology

3.1 Data: UTIn3D

UTIn3D [39] is a robot navigation dataset gathered in a crowded indoor environment. It gives all the lidar frames for all the sessions collected, their localizations computed by the PointMap algorithm, and the label of each point as ground, permanent, shortT, or longT.

Data Collection has also been explored, in case it is deemed necessary down the line. Specifically, the steps necessary to move the robot via a PS4 controller to collect lidar data as a rosbag for each session have been recorded and can be referenced when needed. A script has also been developed to convert a rosbag of lidar data to a series of ply files of input format required by a downstream ML model. The robot used for data collection is the ClearPath Jackal robot, with a mounted lidar for location detection. The setup is shown in Figure 18.

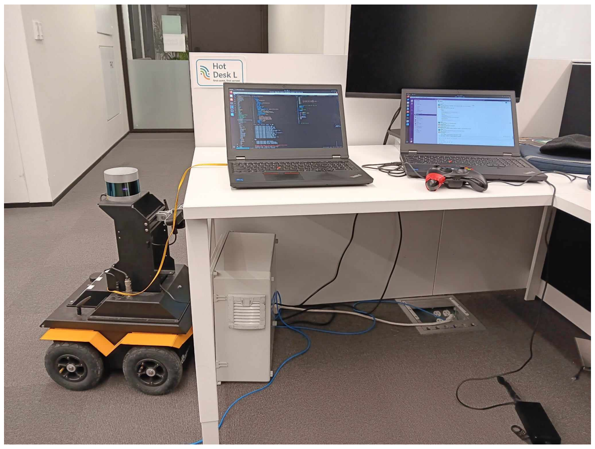


Figure 18: Jackal robot used for data collection.

The file structure of our UTIn3D dataset follows that of Table 2 below. The name of each folder is listed in the left column. The content of each file is summarized in the right column. Importantly, there are two sets of data; UTIn3D_A and UTIn3D_H, collected from UofT Myhal building's Atrium and Hall, respectively. The dataset we use for training combines the two.

File/Folder	Content
annotated_frames	Frames and Point Classifications
calibration	Environment parameters, very light folder
annotation	Map and Trajectory of robot (will need if want to align frames)
runs	(x,y,z) coordinate of each point, required as input to ML model

Table 2: File Structure of UTIn3D Data

3.2 Heuristic Detection Algorithms

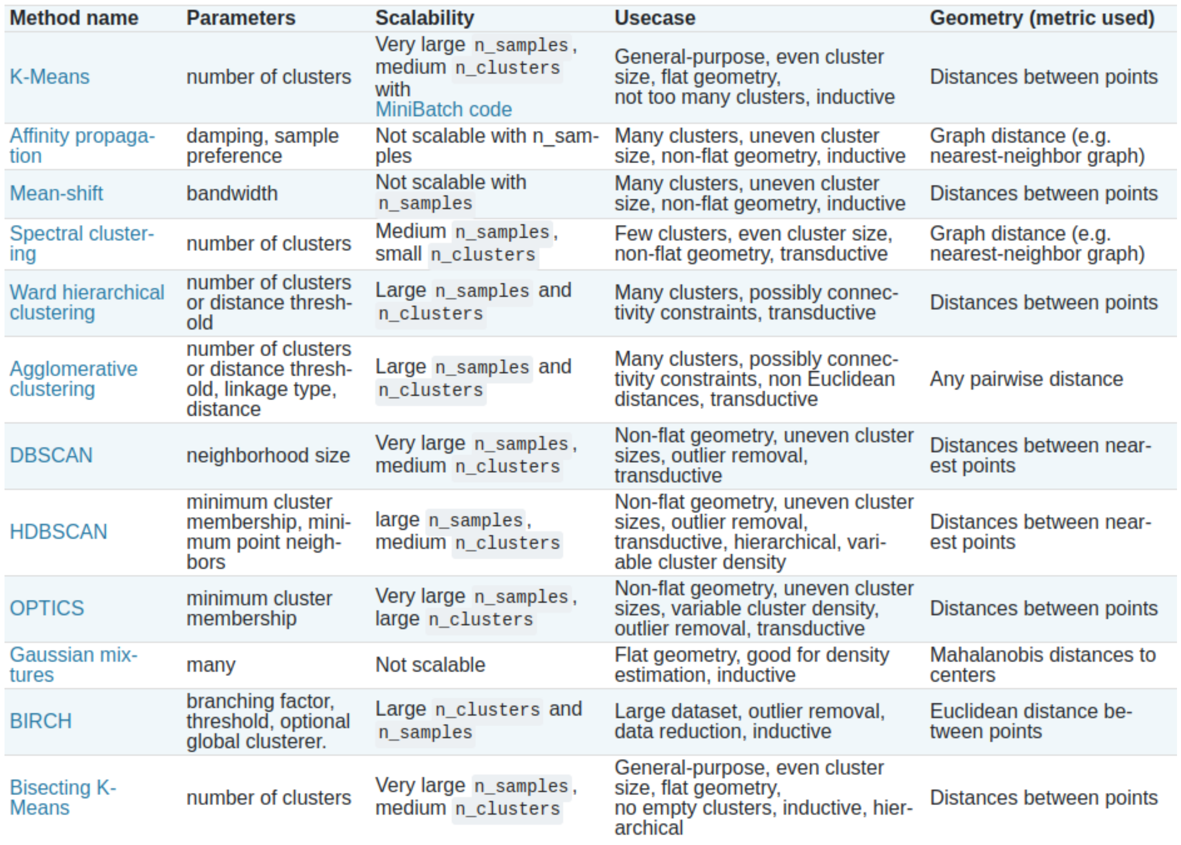
3.2.1 Scikit-learn clustering algorithms

Several heuristic clustering algorithms are covered in Section 2.2. These are summarized in Table 3, where we present the name of the method and a reference to an available implementation. The parameters, usecase, geometry and scalability of several common methods provided by the scikit-learn package are shown in Figure 19. We observe that many of these methods require the number of clusters, we avoid these as the number of people in a scene varies per point cloud.

3.2.2 Why choose DBSCAN?

For our heuristic clustering algorithm, we choose DBSCAN. DBSCAN is robust to noise and outliers; it can effectively filter out noise in data, making it suitable for detecting people even in cluttered environments. DBSCAN allows automatic determination of cluster shapes and sizes- it can identify clusters of various shapes and sizes without prior knowledge, allowing for flexible detection of people in different poses and configurations. Scalability is also good; DBSCAN's time complexity is generally linear with respect to the number of data points, making it efficient for large datasets, which is crucial for real-time people detection applications. Finally, there is no need for predefined number of clusters. Unlike other clustering algorithms, DBSCAN does not require specifying the number of clusters beforehand, making it convenient for detecting variable numbers of people in an image or video frame. Finally, DBSCAN has the ability to handle varying density regions. DBSCAN can detect clusters of different densities, making it suitable for detecting people in crowded areas where density may vary significantly.

In our clustering, we set the DBSCAN parameters to eps=0.5 and min_points=30. The



Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters, inductive	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry, inductive	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry, inductive	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry, transductive	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, transductive	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances, transductive	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes, outlier removal, transductive	Distances between nearest points
HDBSCAN	minimum cluster membership, minimum point neighbors	large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes, outlier removal, transductive, hierarchical, variable cluster density	Distances between nearest points
OPTICS	minimum cluster membership	Very large `n_samples`, large `n_clusters`	Non-flat geometry, uneven cluster sizes, variable cluster density, outlier removal, transductive	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation, inductive	Mahalanobis distances to centers
BIRCH	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction, inductive	Euclidean distance between points
Bisecting K-Means	number of clusters	Very large `n_samples`, medium `n_clusters`	General-purpose, even cluster size, flat geometry, no empty clusters, inductive, hierarchical	Distances between points

Figure 19: Summary of usecases for common clustering algorithms.

parameter eps is the maximum distance between two points for them to be considered close to each other. The parameter min_points is the minimum required number of neighbours within “eps” radius for a point to be added to a cluster.

3.3 End-to-End Network Approaches

3.3.1 Common Point- and Voxel-Based Networks

We are interested in end-to-end point or voxel-based approaches that are publicly available. Our goal is to train one of these on our UTIn3D data after applying the heuristic clustering to derive the ground truth bounding boxes that our algorithm should output. So, the input of our network is a set of N points of the point cloud, and our output is a set of bounding boxes. Table 4 lists some accessible and popular models, as well

Algorithm	**Implementation**
K-Means Clustering	Scikit-learn Implementation
Euclidean Cluster Extraction	Official Python Implementation
DBSCAN	Scikit-learn Implementation
Mean Shift	Scikit-learn Implementation
Method in Section 2.2.3	ROS-based C++ Version
Scan-line Run	C++ Implementation
Depth Clustering	C++ Implementation

Table 3: Heuristic Object Detection Modules we wish to test

as their implementation. We value a network that is both simple to implement/debug and performs well on common indoor datasets. As we have discussed in Section 2.4.2, we prefer voxel-based over point-based architectures as they are generally better on pedestrian and cyclist detection. Importantly, we initially desire to finetune the model trained on Kitti. However, this gave poor results in early stages, so we pivot our strategy to training from scratch using just UTIn3D.

3.3.2 SECOND: Sparsely Embedded Convolutional Detection

We use the Voxel-based neural network, SECOND: Sparsely Embedded Convolutional Detection [40]. At the time of submission, this method produced state-of-the-art results across all classes for KITTI-based 3D detection [41], while running at 40 fps.

The key contributions of this work, as mentioned in their paper are as follows [40]. First, they implement sparse convolution, thereby greatly increasing the speed of training and inference. In fact, they claim to propose an improved method of sparse convolution that allows it to run faster. Furthermore, they propose a novel angle loss regression approach that demonstrates better orientation regression performance than other methods do. Finally, they introduce a novel data augmentation method for LiDAR-only learning problems that greatly increases the convergence speed and performance. We refer the reader to the original paper [40] for full implementation details. For sake of completeness, we do provide a short summary of the network architecture, loss function, and data augmentations here. Further details regarding the network can be found in Appendix A.

Model	Venue	Code
PointRCNN	CVPR 2019	OpenPCDet
Part-A2-Net	IEEE 2019	
PV-RCNN	CVPR 2020	
Voxel R-CNN	AAAI 2021	
MPPNet	ECCV 2022	
PV-RCNN++	IJCV 2023	
VoteNet	ICCV 2019	mmdetection3d
H3DNet	ECCV 2020	
Group-Free-3D	ICCV 2021	
FCAF3D	EECV 2022	
TR3D	Arxiv 2023	
Complex YOLOv4	ECCV 2018	PyTorch
SFA3D	CVPR 2020	PyTorch
VoxelNeXt	CVPR 2023	Python
PillarNeXt	CVPR 2023	Python

Table 4: Popular Architectures for 3D object detection from lidar point cloud

Network Architecture The SECOND network, shown in Figure 20, consists of 3 components: a voxelwise feature encoder, a sparse convolutional middle layer and a Region Proposal Network (RPN).

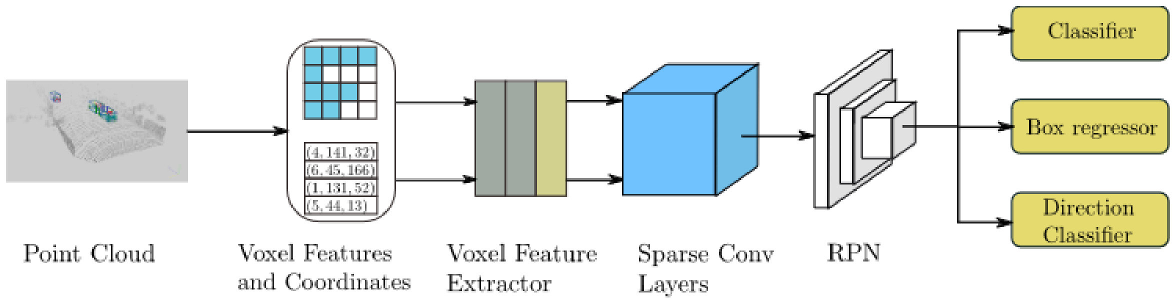


Figure 20: Structure of the SECOND object detector. The input to the detector is a raw point cloud. The network converts this to voxel features and coordinates, and applies two voxel feature encoding layers and a linear layer. Then, they apply a sparse CNN, followed by a RPN to generate the detection.

<ins>Point Cloud Grouping:</ins> The SECOND network starts by voxelizing the input. To do

this, they first allocate a buffer of defined size, as determined by the range of the point cloud and the dimensions of each voxel. The data structure used to store the points for each voxel is a hash table, where keys are voxels and values are points in the voxel. In our experiments, we change the point cloud range to be voxelized from the default to [x_min, y_min, z_min, x_max, y_max, z_max] = [-106, -130, -7, 60, 140, 7], and the voxel size has been changed from [0.1, 0.1, 0.15] to [0.10375, 0.16875, 0.35]. The original model dealt with car, pedestrian and cycle detection; however, we only deal with pedestrian detection. Importantly, the maximum number of points in each voxel is set to 45. This is because pedestrians are small relative to cars, so in the original work, they needed more points for people over cars for the feature extraction.

Voxelwise Feature Encoding: Obtaining voxelwise feature vectors is done with a voxel feature encoding layer (VFE), as introduced by the authors of VoxelNet [37]. A VFE layer takes all points in a voxel and uses a fully connected network (FCN) layer to extract pointwise features. The FCN here is just a linear layer, followed by batch normalization and then ReLU activation. After obtaining pointwise features, elementwise max pooling is used to get locally aggregated features for each voxel. Finally, it tiles the obtained features and concatenates the tiled features and pointwise features. The entirety of the voxelwise feature extractor consists of many of these VFE layers and a FCN layer at the end.

Sparse Convolutional Middle Extractor: The purpose of the *sparse convolutional middle extractor* is to learn information about the z axis and change the input representation from sparse 3D data into a 2D Bird's Eye View (BEV) image. Figure 21 shows the structure of this component. It consists of two phases of sparse convolution (each consisting of several submanifold convolution layers and a normal sparse convolution), which performs downsampling in the z-axis. After the z dimension has been downsampled to one or two, the sparse matrices are converted to dense feature maps. These are then reshaped to 2D BEV image data.

Here, we omit the explanation of sparse convolutional algorithm, in order to avoid too much mathematical complexity that is not directly relevant to our main question. We refer the reader to the first paper to introduce spatially sparse convolutions [42].

Region Proposal Network: Region Proposal Networks [43] are becoming a common part of many detection frameworks. The SECOND architecture uses a single shot multibox detector architecture [44] as the RPN architecture. The input to the RPN consists

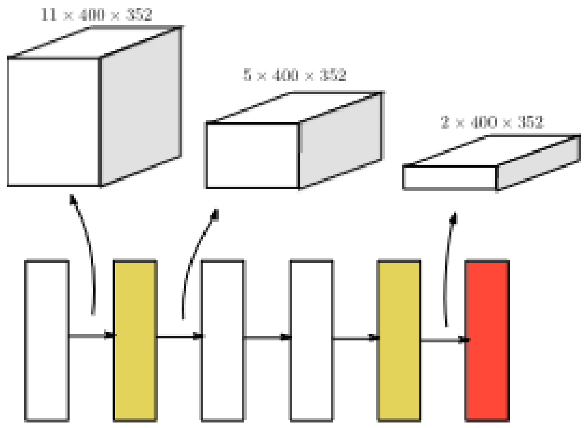


Figure 21: Sparse convolutional middle extractor. Yellow boxes represent sparse convolution, white boxes submanifold convolution and red boxes sparse to dense layers. Upper part of figure shows dimensions of sparse data.

of the output feature maps of the sparse convolutional middle layer. The RPN has 3 stages. Each stage begins with a downsampling convolutional layer, followed by ordinary convolutions. The intermediate downsampled layers are upsampled to feature maps of identical size and then concatenated into one feature map. Finally, three 1x1 convolutions are applied for predicting class, regression offsets and direction.

<u>Anchors and Targets:</u> Anchors and targets are used to match potential bounding boxes with known anchors that represent what we expect people to look like. Because people are approximately the same size, regardless of the dataset, we keep the default anchor parameters. Specifically, for pedestrians, we use anchor dimensions width = 0.6m, length = 0.8m and height = 1.73m.

Each anchor is assigned a one hot vector as the classification target (here, irrelevant as the only class is the person class) and a length 7 vector of box dimension regression targets. We use a threshold method to determine matching and nonmatching classes. Anchors with an Intersection over Union (IoU) measurement over 0.5 are considered matching. Anchors with IoU below 0.35 are considered non matching. Anchors with IoU in between are ignored during training.

Loss: The multitask loss used is a combination of the (1) classification loss, (2) regression loss for box location and dimension, (3) regression loss for angle/orientation and (4) the direction classification loss:

$$L_{total} = \beta_1 * L_{cls} + \beta_2 * (L_{reg-\theta} + L_{reg-other}) + \beta_3 * L_{dir}$$

The classification loss, L_{cls}, is related to how well the network can classify a bounding box as a person, cyclist or car. In our case, it is not too meaningful as we only have one class. $L_{reg-other}$ is the regression loss for the center and {length, width, height} dimensions of the bounding box. $L_{reg-\theta}$ is the novel angle loss regression they use. Finally, L_{dir} is the direction classifier loss, which accounts for the fact that their angle loss treats boxes with opposite directions as being same. The direction classfier uses a softmax loss function.

Novel Angle Loss: Although other terms in the loss function are not new, the angle loss used by SECOND is novel. They use a sine-error loss for angle regression:

$$L_\theta = SmoothL1(sin(\theta_p - \theta_t))$$

This loss solves an adversarial example that VoxelNet fails to address, namely that when the angle is 0 or π, the box stays the same, but normally the loss is high when one is misidentified for the other. It also naturally models intersection over union against the angle offset function.

Data Augmentations: The 3 data augmentation techniques used are *sampling ground truths from the database*, *object noise* and *global rotations and scaling. Sampling ground truth boxes* tackles the problem of too few ground truths during training, which slows convergence and was reported to have hindered performance. To address this, the authors randomly sample ground truth boxes, and incorporate the bounding box and points inside of them in new point cloud frames. During this process, they make sure to avoid collisions with existing boxes. This allows them to increase the number of ground truth boxes per point cloud. Since a person can generally be placed into any free position in an indoor environment, this approach does not affect the nature of the data. *Object noise* deploys the same strategy as VoxelNet, whereby each ground truth and its point cloud are independently randomly transformed. Random rotations are sampled from a uniform distribution [-π/2, π/2], and random transformations from the normal gaussian distribution. Finally, *global rotation and scaling* was applied to the whole point cloud, including all ground truth bounding boxes. Scaling noise was randomly sampled from uniform distribution [0.95, 1.05] and global rotation noise sampled from [-π/4, π/4].

Optimization: The SECOND detector was trained using stochastic gradient descent. The optimizer used was the widely recognized Adam optimizer. The initial learning rate was 0.0002, exponential decay factor was set to 0.8, and the learning rate was decayed every 15 epochs. A decay weight of 0.0001 was used, beta1 value of 0.9 and beta2 value of 0.999. The authors of SECOND used 3 point clouds per minibatch, and trained the Kitti object detection algorithm in 9 hours after running training for 160 epochs (200k iterations). In our case, we ran with identical Adam optimizer parameters, used a Tesla V100 GPU, and ran for only 20 epochs (due to time constraint.) Further experiments will test larger epochs.

4 Findings

4.1 Heuristic Approach

For method 2, we trained the KPConv network with all default parameters on a train split of UTIn3D. Then we performed inference on a hold out test set. The inference gave us predicted dynamic points, after which we applied the heuristic person detector. Some results are shown in Figures 22 and 23. Here, each frame shows points predicted to be dynamic by the inference pipeline. Each colour corresponds to a separate cluster from DBSCAN. Bounding boxes around each cluster are shown.

Qualitatively, the results are quite impressive, but not perfect. We observe that the network does a good job at identifying the people points. However, there are 2 concerns. First, there are many outliers in some of the predictions, which result in spurious clusters. Second, the network sometimes classifies chair points as people. This may not be an error in the network, but rather, in our assumption that the only short term movable objects are people. Indeed, if the chair in this scene is moved around a lot, it may be classified as shortT. We also observe that the clustering algorithm is quite successful at identifying relevant clusters. However, it commonly places outliers as one box. In Section 5, we discuss ways to improve upon these results.

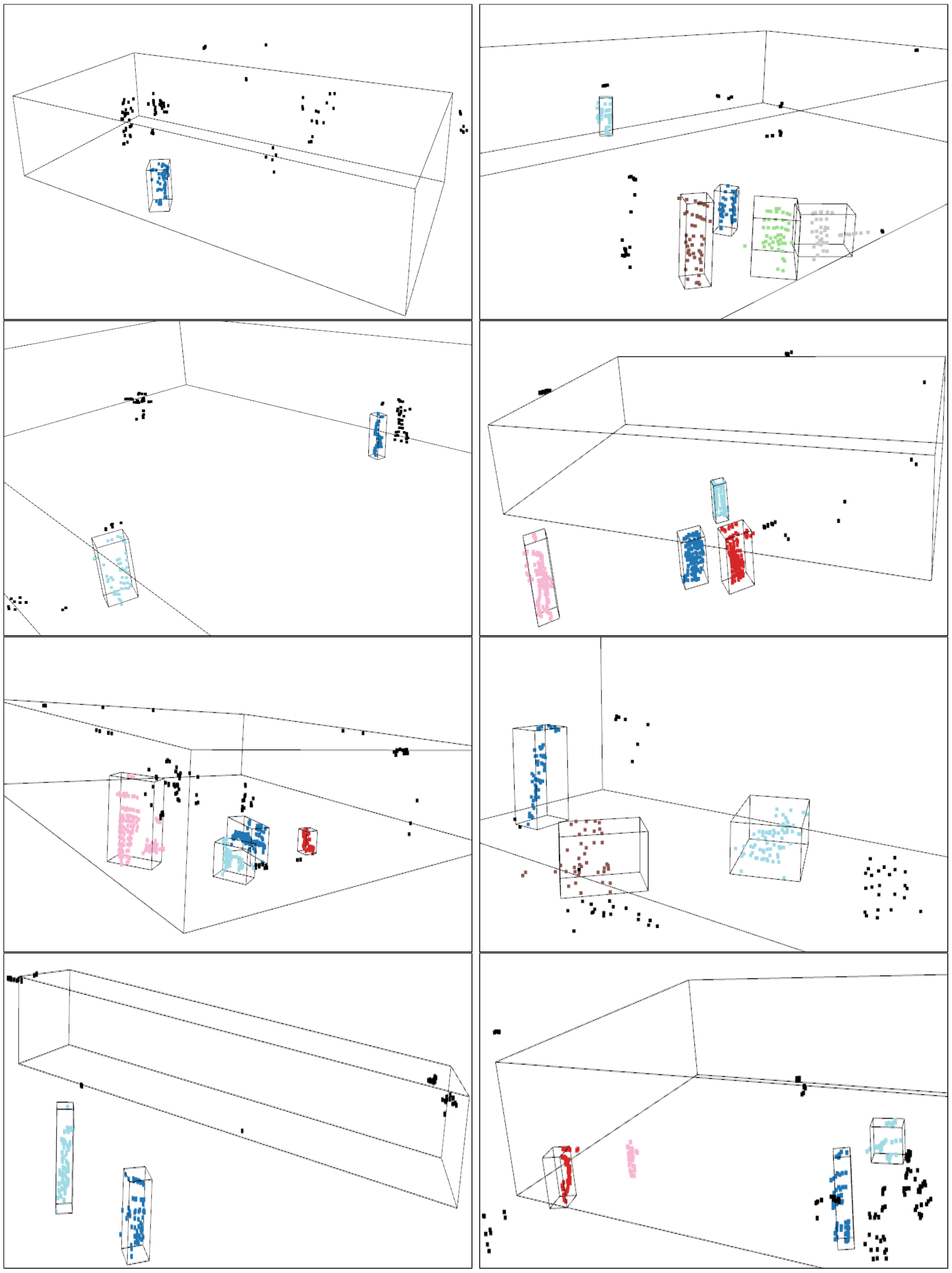


Figure 22: Method 2 sample results 1-8

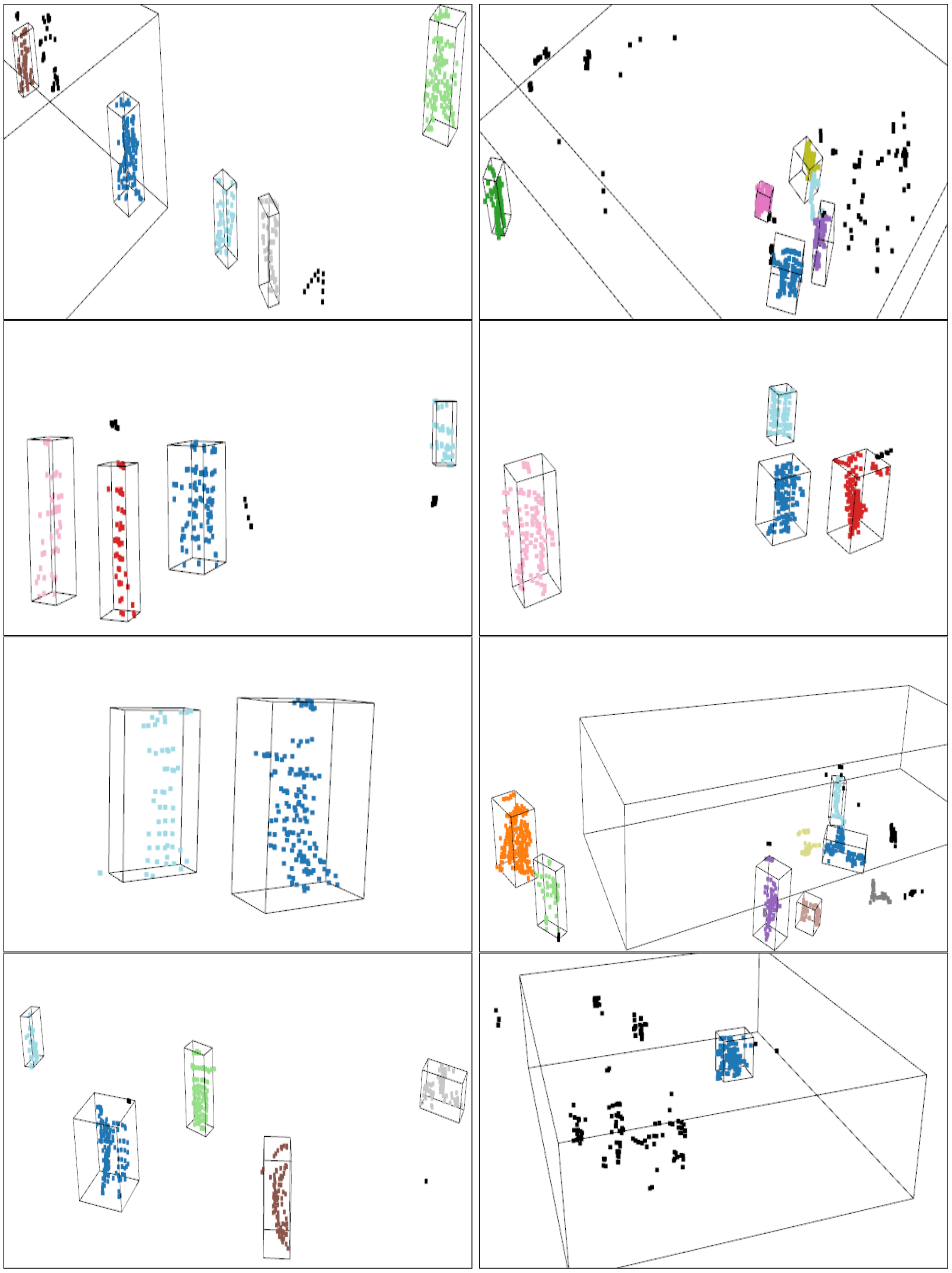


Figure 23: Method 2 sample results 9-16

4.2 Neural Method

Train on single point cloud: In order to train the neural network, we use the SECOND architecture implementation available through OpenPCDet's github repository. We set up the codebase to work both locally and on the remote obelisk machine. The result of training on a single file and testing on that same file is shown in Figure 24. This suggests that our pipeline setup is correct, and we can go ahead with full training.

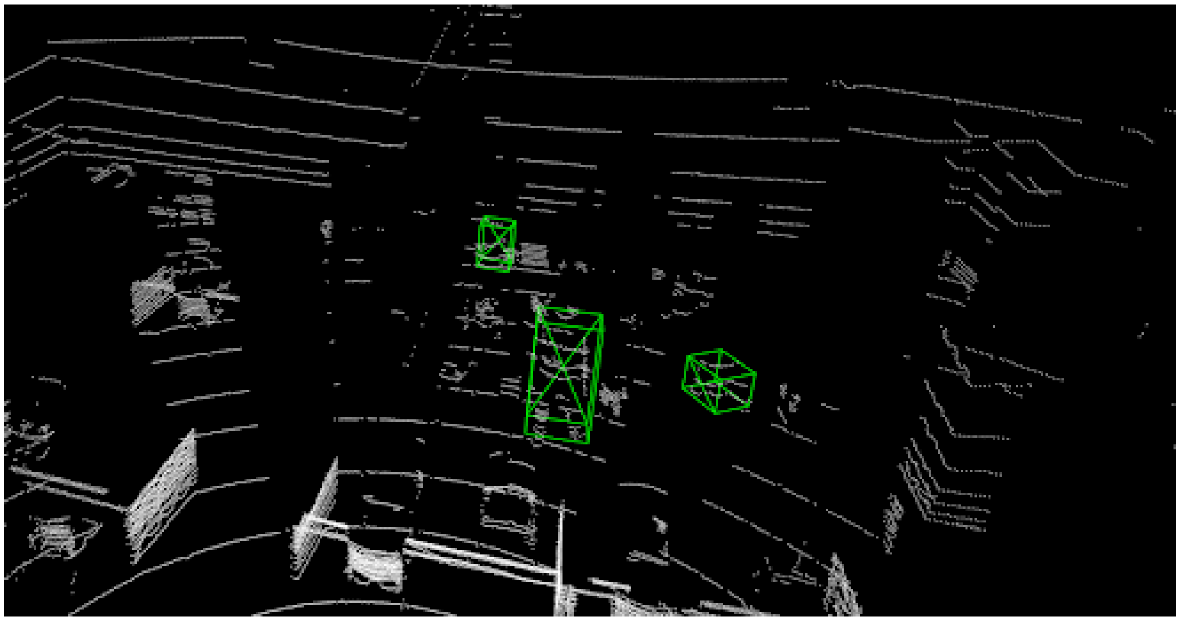


Figure 24: Result of training on one file and testing on same file.

Train on Full UTIn3D dataset: The results in Figure 25 show the output of a single validation file after training the SECOND network with default parameters. From left to right, the outputs show results for num_epochs = 10, 20. Clearly, the neural method does not perform well at this time. However, it is important to note that right now we are using default hyperparameters, and changing these may help us improve results. Furthermore, the existence of spurious clusters in the train dataset (which would have been created by applying heuristic algorithm on slam classification results) will definitely hinder network performance as it learns to predict boxes around outlier bounding boxes.

The ground truth sampling data augmentation strategy may exacerbate these errors. Specifically, it may result in sampling of bounding boxes around outlier data points in different point clouds during training, and the network will have more exposure to learn these incorrect results.

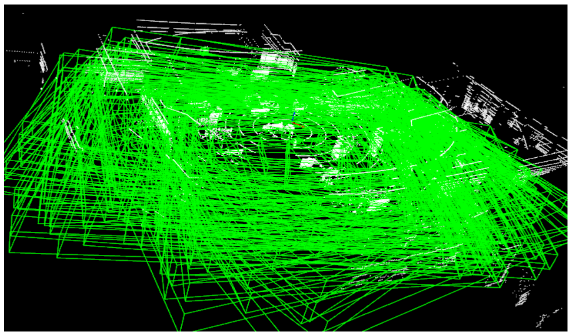
The network does not learn to produce accurate bounding boxes. Upon further inspection, as shown in Appendix B, there are many incorrect bounding boxes. However, we do realize some common patterns. First, many point clouds have bounding boxes that span the entire point cloud, or almost all of it. This is likely due to the existence of bounding boxes around all outlier points, which cover the entire point cloud. These boxes are prevalent in the ground truth dataset, and they are seen here many times per point cloud, perhaps due to the ground truth augmentation strategy mentioned in Section 3.3.2. Second, there are often boxes randomly placed outside of the vicinity of the scene. Again, we suspect they are capturing outlier bounding boxes. Another observation is that most of the boxes are either covering the entire scene, covering outliers away from the scene, or located at the very center of the scene. The center ones are likely the ones that correspond to ”correct” bounding boxes. As of right now, we do not have good results even for the middle boxes. However, we have not time to sufficiently train the network beyond 20 epochs, and it may be useful to set a higher initial learning rate.

5 Future Work

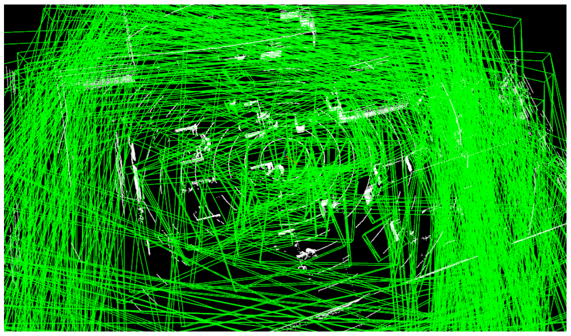
We have made significant progress towards answering the original question, which asked which of the 2 methods work better. We see that at the current iteration of work, the heuristic approach outperforms the end-to-end network. This result is evident qualitatively, so there is no need for quantitative metrics at this time.

There are many future endeavors to make this work more exciting, and they raise the potential for narrowing the gap between the heuristic and neural methods.

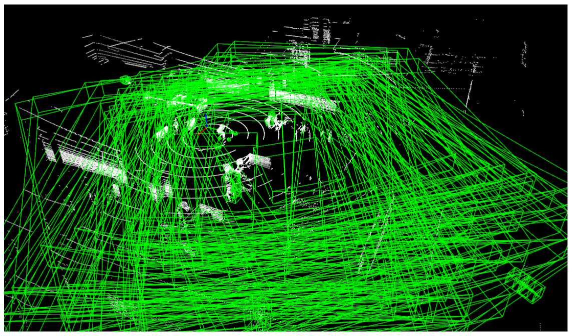
For method 2, we wish to explore different DBSCAN parameters. Right now, the eps and min_points parameters are set to values that work best across a few examples. There is no rigorous method employed for the selection of these parameters. We also wish to implement variations of the DBSCAN algorithm, like HDBSCAN [45] and the strategy mentioned in Section 2.2.3. Another improvement would be the removal of outlier points before clustering. This would avoid the grouping of outlier points as one



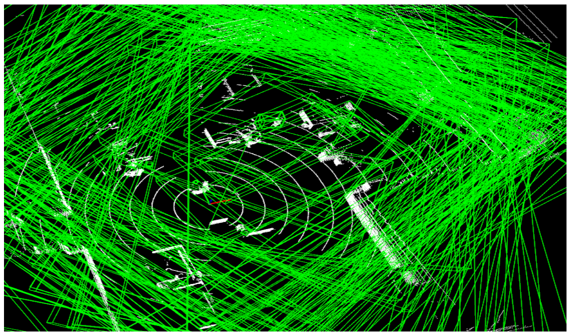
(a) Epoch 10, Sample 1



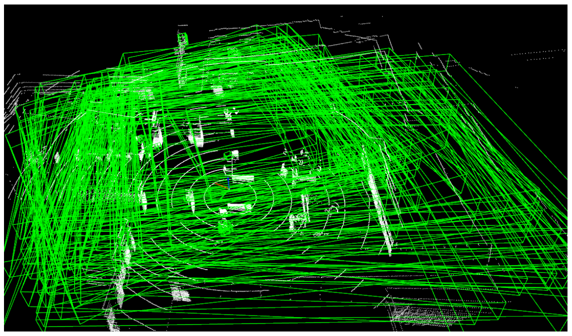
(b) Epoch 20, Sample 1



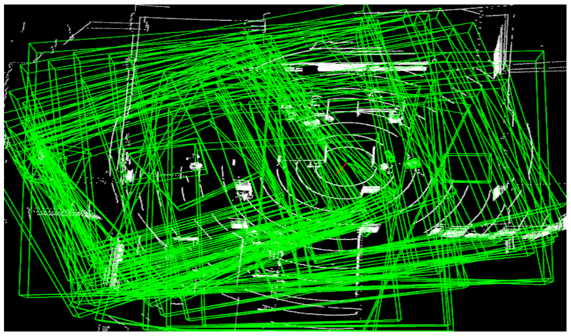
(c) Epoch 10, Sample 2



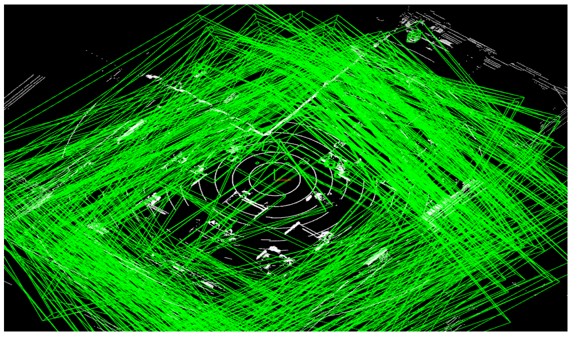
(d) Epoch 20, Sample 2



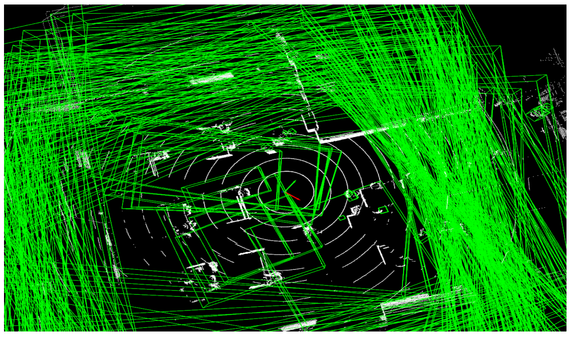
(e) Epoch 10, Sample 3



(f) Epoch 20, Sample 3



(g) Epoch 10, Sample 4



(h) Epoch 20, Sample 4

Figure 25: Results on a single validation point cloud after training network for different number of epochs, with default hyperparameters.

big cluster. This would require a review of strategies for outlier removal, which is a well-studied problem in data science [46].

For method 1, our immediate strategy for improvement is hyperparameter tuning. Thus far, we have only experimented with one hyperparameter setting (the default). However, we are interested in exploring different learning rates, voxel sizes, anchor sizes and anchor rotations, amongst other hyperparameters. We suspect that the learning rate will have an impact on training speed and convergence. Since we start training from scratch, it may be better for us to start with a larger learning rate than the default, and decay the learning rate slower than default. Regarding voxel sizes, we suspect that density differences between our data and the Kitti dataset may render this parameter important. Since the anchor size and rotations for pedestrians should not differ significantly across datasets, we don't suspect experimentation with anchor sizes and anchor rotations will have much effect. However, we include them as hyperparameters of interest due to their importance to SECOND's people detection strategy.

Continuing, we wish to experiment with dataset size and train time augmentations. Regarding dataset size, we wish to investigate the impact of simulated data on performance - both for training the KPConv network and for training the SECOND network. Regarding train time augmentations, we wish to implement the augmentations boasted about by the authors of SECOND (currently, flag is set to false).

Besides comparing heuristics and neural approaches via the investigation of the SECOND architecture, we also wish to compare different neural methods. The SECOND architecture is state-of-the-art for Kitti object detection, but we wish to test methods that are SOTA for other datasets. Moreover, SECOND performs better on outdoor datasets than indoor. Other, newer architectures may be more suited for point cloud object detection in indoor cluttered environments. Expanding, it is interesting to investigate different classes of method. SECOND is a voxel-based method; future experiments would test point-based, mixed point and voxel based, and range-based as well.

Once the steps above have been implemented, the next stage is to compare qualitatively. If it happens that the heuristic method clearly outperforms the neural method, then we will select it for implementation in the Jackal. Otherwise, we perform a quantitative comparison across the validation set, and select the method that does better across relevant metrics, weighted by importance. Performance metrics include intersection over union, accuracy, precision and recall.

There are methods to expand the scope of this work that give the potential for further improvement. Most notably, we ideally want to use multiple frames instead of just one. The time series nature of the data should allow us to refine the bounding boxes; this is because with more frames, we can increase our confidence in what is and isn't a person. Another notable expansion of scope involves our assumption that all dynamic points are people points. In a general real-world setting, we will not be restricted to a people-chair-table environment. Thus, our people detector would really just become a dynamic object detector. We would require either a cluster classification (see section 2.3) module on top, or pivot to a different human detection strategy.

Another important extension is dealing with occlusions and interacting objects. Once we have a minimum viable product, we wish to add components to our network to distinguish between objects when two or more objects are interacting; for example, a person and chair, as well as one object occluding another object; for example, one person walking in front of another.

6 Conclusion

To conclude, the heuristic method (method 2) works better than the current iteration of the end-to-end neural method (method 1). For the heuristic method, the clustering heuristic does a good job, except for placing bounding boxes around outlier points. To solve this, we can filter out incorrect bounding boxes by center location and dimensions. For example, it is unrealistic that a person to big enough to require the entire scene as a bounding box, so these boxes would be filtered out. For the neural approach, it is possible to produce a ground truth dataset that is more pure by filtering out outlier bounding boxes. Note that it is also possible to filter out spurious bounding boxes during inference (instead of train), but this is less desirable. Once we create the "purer" dataset, we hope it gives better performance after training. If it does not, then we perform a thorough search of the hyperparameter space, loss function, data augmentations and other variable factors. We are certain that the neural approach can give better results than the current iteration through further training and hyperparameter tuning. However, so long as our assumption that people points are the only dynamic points hold, it is hard to see the neural method beat the heuristic method, as KPConv's task of classifying each point into 4 classes is more well-defined than SECOND's task of proposing bounding boxes, which relies on more components (e.g., region proposal network).

References

[1] G. Angelopoulos, N. Baras, and M. Dasygenis, “Secure autonomous cloud brained humanoid robot assisting rescuers in hazardous environments,” *Electronics*, vol. 10, no. 2, p. 124, 2021.

[2] Y. Chen, C. Yang, B. Song, N. Gonzalez, Y. Gu, and B. Hu, “Effects of autonomous mobile robots on human mental workload and system productivity in smart warehouses: A preliminary study,” in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, SAGE Publications Sage CA: Los Angeles, CA, vol. 64, 2020, pp. 1691–1695.

[3] M. Imad, O. Doukhi, and D.-J. Lee, “Transfer learning based semantic segmentation for 3d object detection from point cloud,” *Sensors*, vol. 21, no. 12, p. 3964, 2021.

[4] A. Azim and O. Aycard, “Detection, classification and tracking of moving objects in a 3d environment,” in *2012 IEEE Intelligent Vehicles Symposium*, IEEE, 2012, pp. 802–807.

[5] D. Brščić, T. Kanda, T. Ikeda, and T. Miyashita, “Person tracking in large public spaces using 3-d range sensors,” *IEEE Transactions on Human-Machine Systems*, vol. 43, no. 6, pp. 522–534, 2013.

[6] W. Chen, Y. Zhu, Z. Tian, F. Zhang, and M. Yao, “Occlusion and multi-scale pedestrian detection a review,” *Array*, p. 100 318, 2023.

[7] L. E. Navarro-Serment, C. Mertz, and M. Hebert, “Pedestrian detection and tracking using three-dimensional ladar data,” *The International Journal of Robotics Research*, vol. 29, no. 12, pp. 1516–1528, 2010.

[8] K. Kidono, T. Miyasaka, A. Watanabe, T. Naito, and J. Miura, “Pedestrian recognition using high-definition lidar,” in *2011 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2011, pp. 405–410.

[9] H. Wang, B. Wang, B. Liu, X. Meng, and G. Yang, “Pedestrian recognition and tracking using 3d lidar for autonomous vehicle,” *Robotics and Autonomous Systems*, vol. 88, pp. 71–78, 2017.

[10] K. Babacan, L. Chen, and G. Sohn, “Semantic segmentation of indoor point clouds using convolutional neural network,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 4, pp. 101–108, 2017.

[11] H. Thomas, B. Agro, M. Gridseth, J. Zhang, and T. D. Barfoot, “Self-supervised learning of lidar segmentation for autonomous indoor navigation,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021.

[12] Z. Yan, L. Sun, T. Duckctr, and N. Bellotto, “Multisensor online transfer learning for 3d lidar-based human detection with a mobile robot,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 7635–7640.

[13] Z. Yan, T. Duckett, and N. Bellotto, “Online learning for 3d lidar-based human detection: Experimental analysis of point cloud clustering and classification methods,” *Autonomous Robots*, vol. 44, pp. 147–164, 2020.

[14] C. Romero-González, A. Villena, D. González-Medina, J. Martínez-Gómez, L. Rodríguez-Ruiz, and I. García-Varea, “Inlida: A 3d lidar dataset for people detection and tracking in indoor environments,” in *International Conference on Computer Vision Theory and Applications*, SCITEPRESS, vol. 7, 2017, pp. 484–491.

[15] H.-L. Tang, S.-C. Chien, W.-H. Cheng, Y.-Y. Chen, and K.-L. Hua, “Multi-cue pedestrian detection from 3d point cloud data,” in *2017 IEEE international conference on multimedia and expo (ICME)*, IEEE, 2017, pp. 1279–1284.

[16] D. Tiozzo Fasiolo, E. Maset, L. Scalera, S. Macaulay, A. Gasparetto, and A. Fusiello, “Combining lidar slam and deep learning-based people detection for autonomous indoor mapping in a crowded environment,” *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 43, pp. 447–452, 2022.

[17] J.-S. Yoon, S.-H. Bae, and T.-y. Kuc, “Human recognition and tracking in narrow indoor environment using 3d lidar sensor,” in *2020 20th International Conference on Control, Automation and Systems (ICCAS)*, IEEE, 2020, pp. 978–981.

[18] Ò. Lorente, J. R. Casas, S. Royo, and I. Caminal, “Pedestrian detection in 3d point clouds using deep neural networks,” *arXiv preprint arXiv:2105.01151*, 2021.

[19] C. M. Sánchez, J. Capitán, M. Zella, and P. J. Marrón, "Point-cloud fast filter for people detection with indoor service robots," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, IEEE, 2020, pp. 161–165.

[20] F. Pomerleau, F. Colas, Roland, Siegwart, Stéphane, and Magnenat, "Comparing icp variants on real-world data sets," 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:261299591.

[21] F. Pomerleau, P. Krüsi, F. Colas, P. T. Furgale, and R. Y. Siegwart, "Long-term 3d map maintenance in dynamic environments," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3712–3719, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:11211222.

[22] Z. Yan, T. Duckett, and N. Bellotto, "Online learning for 3d lidar-based human detection: Experimental analysis of point cloud clustering and classification methods," *Autonomous Robots*, vol. 44, pp. 147–164, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:201894703.

[23] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," *KI - Künstliche Intelligenz*, vol. 24, pp. 345–348, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:3345029.

[24] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Knowledge Discovery and Data Mining*, 1996. [Online]. Available: https://api.semanticscholar.org/CorpusID:355163.

[25] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 603–619, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:691081.

[26] D. Zermas, I. Izzat, and N. Papanikolopoulos, "Fast segmentation of 3d point clouds: A paradigm on lidar data for autonomous vehicle applications," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5067–5073. DOI: 10.1109/ICRA.2017.7989591.

[27] I. Bogoslavskyi and C. Stachniss, "Efficient online segmentation for sparse 3d laser scans," *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, vol. 85, pp. 41–52, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:65118079.

[28] M. Häselich, B. Jobgen, N. Wojke, J. Hedrich, and D. Paulus, "Confidence-based pedestrian tracking in unstructured environments using 3d laser distance measurements," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4118–4123, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:10657230.

[29] K. Li, X. Wang, Y. Xu, and J. Wang, "Density enhancement-based long-range pedestrian detection using 3-d range data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 1368–1380, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:7810729.

[30] D. Z. Wang and I. Posner, "Voting for voting in online point cloud object detection," in *Robotics: Science and Systems*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:15568286.

[31] M. Deuge, A. Quadros, C. Hung, and B. Douillard, "Unsupervised feature learning for classification of outdoor 3d scans," *Australasian Conference on Robotics and Automation, ACRA*, Jan. 2013.

[32] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4338–4364, 2021.

[33] A. Teichman and S. Thrun, "Tracking-based semi-supervised learning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 804–818, 2012.

[34] J. Shackleton, B. VanVoorst, and J. Hesch, "Tracking people with a 360-degree lidar," in *2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance*, IEEE, 2010, pp. 420–426.

[35] J. Mao, S. Shi, X. Wang, and H. Li, "3d object detection for autonomous driving: A review and new outlooks," *arXiv preprint arXiv:2206.09474*, 2022.

[36] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[37] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4490–4499.

[38] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:10328909.

[39] H. Thomas, M. G. d. S. Aurin, J. Zhang, and T. D. Barfoot, "Learning spatiotemporal occupancy grid maps for lifelong navigation in dynamic scenes," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2022.

[40] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, 2018.

[41] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[42] B. Graham, "Spatially-sparse convolutional neural networks," *arXiv preprint arXiv:1409.6070*, 2014.

[43] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[44] W. Liu, D. Anguelov, D. Erhan, *et al.*, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, Springer, 2016, pp. 21–37.

[45] L. McInnes and J. Healy, "Accelerated hierarchical density based clustering," in *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*, IEEE, 2017, pp. 33–42.

[46] M.-J. Rakotosaona, V. L. Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov, "Pointcleannet: Learning to denoise and remove outliers from dense point clouds," *Computer Graphics Forum*, vol. 39, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:57572894.

A Network Details

The following is a description of the network details of SECOND, copied over exactly from the original paper. Note that we are only interested in pedestrian detection, not car or cyclist.

"We propose the use of two networks: a large one and a small one. Points that lie outside the camera view frustum need to be removed.

For car detection, two VFE layers are used in SECOND, namely, VFE(32) and VFE(128) for the large network and VFE(32) and VFE(64) for the smaller network, following a Linear(128) layer. Thus, the dimensions of the output sparse tensor are 128 × 10 × 400 × 352 for the large network and 128 × 10 × 320 × 264 for the small network. Then, we use a two-stage sparse CNN for feature extraction and dimension reduction, as shown in Figure 3. Each convolutional layer follows a BatchNorm layer and a ReLU layer. All sparse convolutional layers have a 64-output feature map, a kernel size of (3, 1, 1) kernel size and a stride of (2, 1, 1). The dimensions of the output of the middle block are 64 × 2 × 400 × 352 for the large network. Once the output has been reshaped to 128 × 400 × 352, the RPN network can be applied. Figure 4 shows the architecture of the RPN. We use Conv2D(cout, k, s) to represent a Conv2D-BatchNorm-ReLU layer and DeConv2D(cout, k, s) to represent a DeConv2D-BatchNorm-ReLU layer, where cout is the number of output channels, k is the kernel size and s is the stride. Because all layers have the same size across all dimensions, we use scalar values for k and s. All Conv2D layers have the same padding, and all DeConv2D layers have zero padding. In the first stage of our RPN, three Conv2D(128, 3, 1(2)) layers are applied. Then, five Conv2D(128, 3, 1(2)) layers and five Conv2D(256, 3, 1(2)) layers are applied in the second and third stages, respectively. In each stage, s = 2 only for the first convolutional layer; otherwise, s = 1. We apply a single DeConv2D(128, 3, s) layer for the last convolution in each stage, with s = 1, 2, and 4 for the three stages, sequentially. For pedestrian and cyclist detection, the only difference with respect to car detection is that the stride of the first convolutional layer in the RPN is 1 instead of 2."

The block diagram in Figure 26 below accurately displays the architecture of the RPN, which is a core component of object detection algorithms.

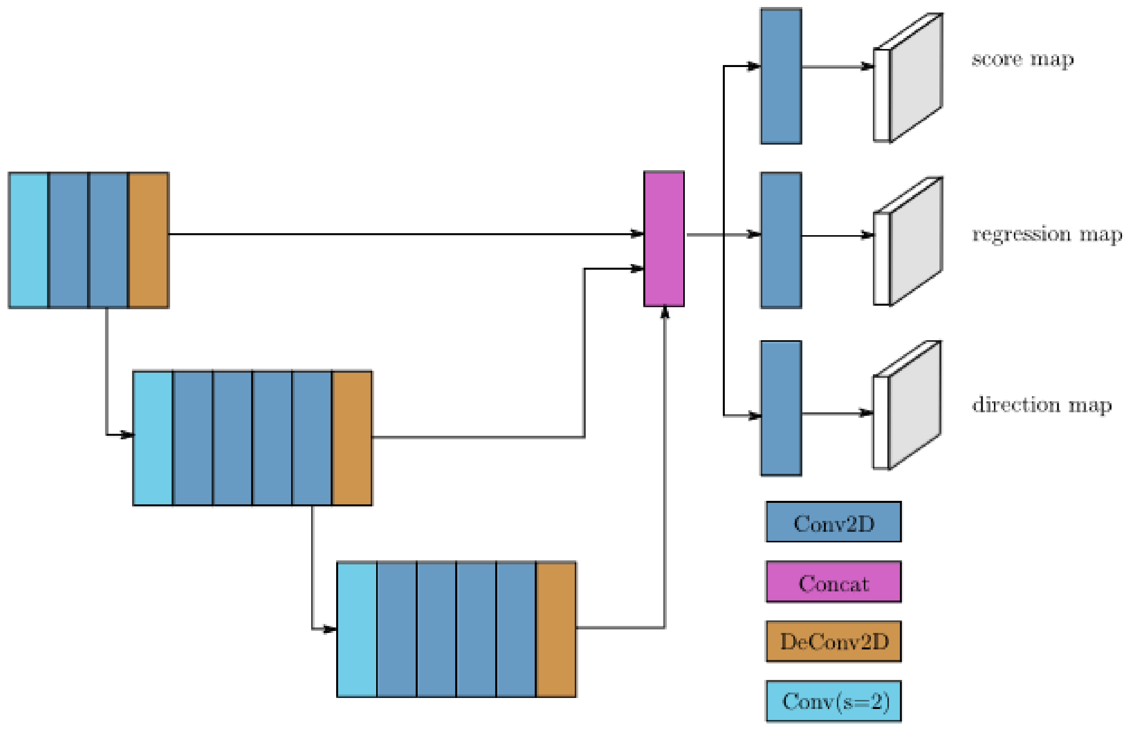


Figure 26: Region Proposal Network architecture. Blue boxes are convolutional layers, purple boxes are concatenation layers, sky blue are stride-2 downsampling convolutional layers and brown boxes are transpose convolutional layers.

B Neural Method Additional Results

We present some more results from method 1 here. The leftmost images show the output of the network. The images to their right show the same output, zoomed in to show relevant details.

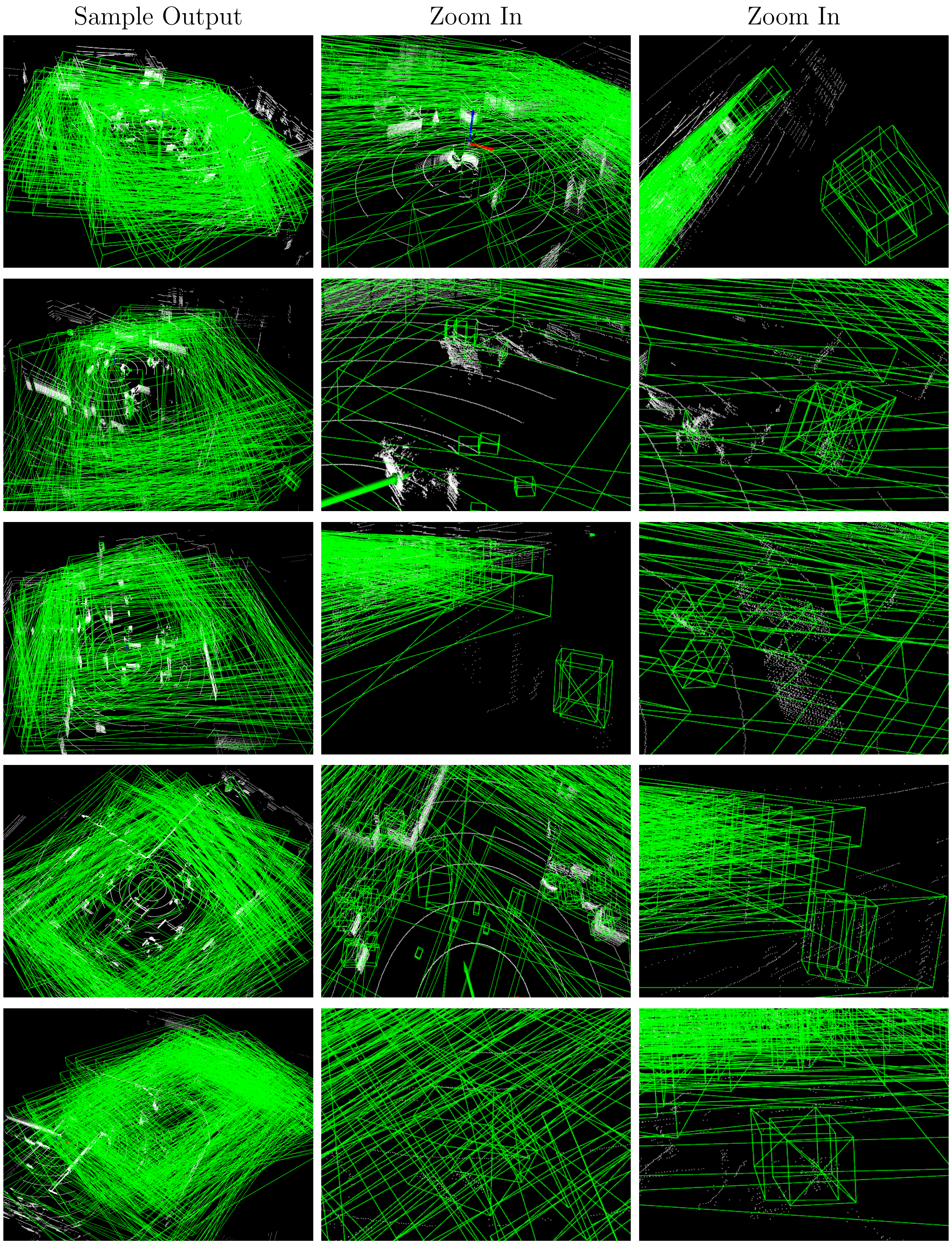


Figure 27: Extra results for method 1. As one can see, it does not perform well.