MASTER'S THESIS 2024

Point-Cloud-Based Crowd Counting

Isak Jakobsson, Jonathan Runeke

ISSN 1650-2884
LU-CS-EX: 2024-30

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-30

Point-Cloud-Based Crowd Counting

Punktmolnsbaserad personräkning

Isak Jakobsson, Jonathan Runeke

Point-Cloud-Based Crowd Counting

Isak Jakobsson
is6427ja-s@student.lth.se

Jonathan Runeke
jo8561ru-s@student.lth.se

June 17, 2024

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Maj Stenmark, maj.stenmark@cs.lth.se

Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

Automating the process of counting people in crowds is crucial for enhancing safety in various settings where crowds are present. Although camera-based crowd-counting methods exist, they often struggle in suboptimal lighting conditions and fail to safeguard the privacy of individuals within the crowds. To address these limitations, we propose the use of LiDAR sensors, which offer the dual advantages of maintaining high accuracy under all lighting conditions and preserving privacy. LiDAR (Light Detection and Ranging) technology uses lasers to scan an environment and generate a point cloud consisting of coordinates in three-dimensional space. In this thesis, we developed and evaluated four methods for crowd counting using LiDAR, three of which were deep-learning-based and one based on a classical machine-learning algorithm. The models were trained on synthetic data generated using the game creation software Unreal Engine 5. All four approaches underwent evaluation based on their performance concerning people quantity, crowd density, and distance from the sensor. Our findings indicate that the classical machine-learning approach, employing the DBSCAN clustering algorithm, outperforms the deep-learning approaches. Although the deep-learning models show promising features, they would benefit from training on higher-quality data. We conclude by recommending further development of point-cloud-based crowd counting, thereby paving the way for a safer future in crowded environments.

Keywords: LiDAR, Point Cloud, Machine Learning, Density Estimation, Crowd Counting, Clustering, Synthetic Data

Acknowledgements

Firstly, we would like to thank our supervisor, Maj Stenmark, for her support and feedback throughout our master's thesis. We also wish to express our gratitude to the team we had the opportunity to work alongside, whose continuous support and positive work environment we especially valued.

Front page image taken by Nazarizal Mohammad, depicting a sunset over Bungkarno Stadium, Jakarta [1].

Contents

Chapter 1
Introduction

Crowd safety and analysis is an important area of research, where crowd counting is an essential component. The ability to automatically count and track people in larger groups is of great interest in applications ranging from ensuring general safety to optimising resource allocations. Recent advancements in technology regarding machine learning and image processing have led to increased performance of camera-based counting methods. However, cameras are not always the optimal choice of sensor. In this master's thesis, Light Detection and Ranging (LiDAR)-based people counting methods were developed and tested to combat the challenges camera-based methods may face.

Crowds may arise for multiple different reasons, where common cases include musical concerts, large queues, evacuations, and public transport. As every crowd is different, flexible and dynamic methods are needed to accurately analyse them. Denser crowds are generally more dangerous and may, in some cases, be deadly. In 2021, ten people lost their lives at a concert in Houston, USA, due to constant pressure from people around them [2]. To help increase safety and reduce the risk of similar tragic events, more advanced crowd analysis methods need to be developed.

In this thesis, four methods for people counting using LiDAR were developed and evaluated. Three of the methods were deep-learning-based, and one employed a more traditional machine-learning algorithm. Popular techniques used on LiDAR data were combined with more traditional techniques used on camera-based methods. All networks have been trained on simulated data generated using the game engine Unreal Engine 5 (UE5).

1.1 Task and Purpose

The primary objective of this master's thesis was to produce an automated method for determining the number of people in a crowd using LiDAR sensors. This involved conducting research on existing methods and subsequently devising our own models based on this research. The models were to be compared to identify the most promising techniques for potential fu-

ture refinement and implementation. Furthermore, this thesis aimed to test the usability of deep learning on point clouds for crowd counting, compared to traditional machine learning without the use of neural networks.

A crucial aim of this research was to enhance crowd safety measures. Accurate crowd size estimation allows for proactive measures to mitigate potential hazards and minimise the risk of harm to individuals in crowded environments. Beyond immediate safety concerns, this research also intends to provide valuable insights for various stakeholders. Accurate crowd analysis holds significance for marketing strategies and urban planning, enabling businesses and authorities to understand crowd behaviour, preferences, and demands. This knowledge can optimise resource allocation and decision-making processes to allow for better planning of future events.

1.1.1 Research Questions

Based on the task above, these research questions could be formulated:

- **RQ1:** How can people counting be effectively implemented using point clouds?
- **RQ1.1:** How does the number of individuals in the crowd affect the accuracy and performance of the people counting system?
- **RQ1.2:** What is the impact of crowd density on the performance of the people counting system?
- **RQ1.3:** How does the distance from the LiDAR sensor influence the system's performance in counting people?
- **RQ2:** Is deep learning a viable approach for solving the task of people counting using LiDAR data?
- **RQ2.1:** How do deep-learning models compare to traditional machine-learning algorithms in terms of accuracy and efficiency for people counting?

By addressing these research questions, this thesis aims to contribute to the field in the following ways:

- **CS1:** Expanding the scope of applications of LiDAR based surveillance systems.
- **CS2:** Provide insights into the performance characteristics of people counting systems under varying conditions.
- **CS3:** Evaluate the effectiveness of deep-learning models trained on synthetically generated point clouds to solve the task of crowd counting.

1.2 Method

In order to be able to answer our research questions, we decided to develop, implement, and evaluate several different approaches. After grasping what was already possible in this field

through a continuous literary study, we were able to begin to work with a specific strategy. Inspiration for other solutions came from unexpected discoveries in previous implementations combined with further understanding of how to use machine learning to solve the task of crowd counting. This resulted in several viable approaches, which we ultimately narrowed down to four, namely, human clustering, head clustering, and two different implementations of density estimation.

Some datasets were needed for this master's thesis. Beyond a training dataset, specific datasets were required to answer **RQ1.1**, **RQ1.2**, and **RQ1.3**. When producing these datasets, two questions arose:

- What data is already available?
- How can we produce more data?

After deciding how to address these questions, we began to train, test, and evaluate our approaches.

1.3 LiDAR

LiDAR is a type of sensor that has been growing in popularity since its invention in the 1960s [3]. The technology has been applied in multiple areas such as agriculture, vehicle industry, and military, and more applications are being explored [4]. In recent years, the autonomous driving industry has found good use of LiDAR to improve distance measuring and positioning. The accuracy and robustness of the sensor, which performs well in suboptimal conditions, make it a good choice for this type of application.

The LiDAR is an active remote sensor, meaning that it sends out a pulse of energy and measures the changes in the returned signal [4]. This is in contrast to passive sensors, such as cameras and thermal cameras, which only collect naturally occurring radiation without emitting energy. Being an active sensor means that it can still perform well in conditions such as darkness and bad weather, however, it also comes with its disadvantages. Depending on the frequency of light used, some materials can be difficult for the LiDAR to accurately measure. For example, water may absorb the light, and mirrors or glass may reflect most of the light.

The main idea behind how the LiDAR works is the reflection of light [4]. Since the speed of light is known, it is possible to measure the time for a beam of light to reflect off a surface, bounce back, and use that to calculate the distance. This is the principle on which LiDAR works. Mirrors are used to direct a laser in a scanning pattern and measure the distance to thousands of points in its field of view. The density of points can be adjusted for specific tasks, a lower number of points means more frames per second that can be captured. From all of these measurements, a point cloud can be constructed for each captured frame.

A point cloud is a representation of LiDAR data, which can be visualised using a computer. It is a cloud-like structure with points scattered for each of the received reflections from the LiDAR scan. The points' distance and angle to the origin of the point cloud are the same distance and angle as from the LiDAR to the reflected surface. An advantage of point clouds over normal images is that they can be interacted with, they do not necessarily have to be viewed from the perspective of the LiDAR. With a stream of LiDAR data, it is also

possible to construct a model of the background by looking at the points that stay the same over a long period of time. Using this model, it is possible to filter out irrelevant parts of the point clouds, making them easier to manage and understand.

Figure 1.1 shows a visualisation of a LiDAR point cloud.

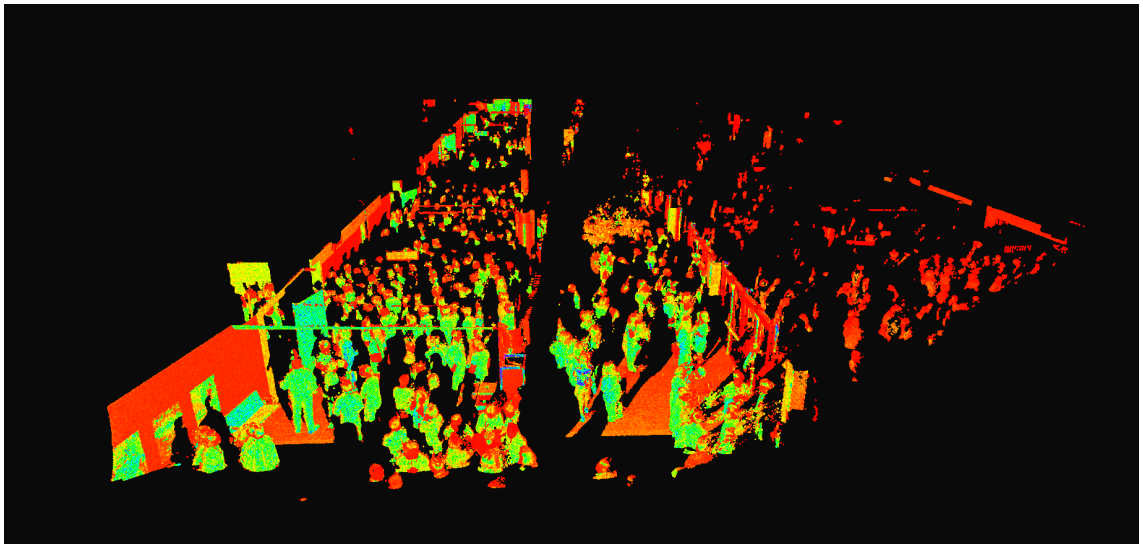


Figure 1.1: A crowd, captured at Malmö Arena, visualised in a point cloud. The colour is a gradient from red to blue, where red points have a low reflection intensity and blue points have a high reflection intensity.

1.4 Limitations and Assumptions

A crucial limitation to this master's thesis was the lack of real annotated LiDAR data. As a large part of the thesis was to evaluate the application of deep learning for crowd counting, large amounts of data were needed to train the models. With the time it takes to annotate point clouds in consideration, the option to automatically generate synthetic data was chosen. However, simulating data came with its own limitations, discussed in subsection 5.1.2. Since data generation was not the main objective of this master's thesis, limited time was spent on it.

Training deep-learning models is highly resource-intensive and requires powerful computers. The computer that was performing the training of the models was equipped with a NVIDIA GeForce RTX 3060 graphics card with 12GB of video memory. Even though this is normally considered a powerful graphics card, it puts a limitation on how large the models can be in terms of trainable parameters as well as how much data can be processed in a single batch.

1.5 Ethics

LiDAR offers more privacy than cameras. As mentioned in section 1.3, point clouds have a low resolution and do not include any colours, therefore, it is almost impossible to recognise people just from LiDAR data. This is especially evident when comparing point clouds to cameras, which generate pictures that resemble the world we are used to seeing with our eyes. To be able to recognise someone, features like facial details, hair, skin colour, and clothes are used. None of these features are noticeable through LiDAR, while all of them exist in high-resolution pictures.

When dealing with security, especially the surveillance of people in public and private situations, it is crucial to adhere to current regulations. One such regulation is the EU Global Data Privacy Regulation (GDPR). LiDAR holds a clear advantage over cameras because it is non-identifying and thus automatically complies with GDPR. However, it should be noted that individuals with very distinct features, such as being exceptionally tall or having two heads, could still be identifiable in LiDAR data.

Injuries and deaths have been the result of multiple crowd accidents in recent history. One of the more serious accidents was the stampede in Houston, USA, where ten people lost their lives and approximately 300 people were injured. The organisers of this event were not expecting this incident, even though they had protocols for bomb or terrorist threats, shootings, and severe weather [2]. This shows the current lack of knowledge in this area and the importance of improved crowd safety.

Malmö Stad has developed a chart for different densities of crowds, using colours to indicate what type of crowd it is. The chart is made up of 5 zones: green zone, blue zone, yellow zone, orange zone, and red zone. The green zone is when everyone has sufficient space to be able to have fully stretched-out arms while spinning, roughly four square metres per person. The blue zone is when everyone has space to stand without touching anyone else. The yellow zone is when people are pressed together but can still wiggle their arms. The orange zone is when the crowd is so compact that you cannot even move your arms anymore. Lastly, the red zone is when some people lose contact with the floor, being lifted by the pressure of the crowd. To counteract the red zone, people have to be lifted and pulled out of the crowd by security guards. The zones are used to detect when crowds are starting to become dangerous.

1.6 Contribution Statement

Throughout the master's thesis, we have been working side by side, always discussing with each other. By working together, it was possible for both of us to get a full understanding of the entire thesis. However, in order to work in parallel, we continuously divided the work between us. An example of that is when we started to implement the machine-learning modules. Here, Jonathan focused on the task of head clustering, while Isak focused on the density estimation approaches.

1.7 Outline

Important theories and concepts behind our work are provided in chapter 2. In chapter 3, we describe how our datasets were made, followed by a description of our four proposed approaches. Then, all the results will be presented in chapter 4 and discussed in chapter 5. chapter 6 provides a conclusion of the master's thesis.

Chapter 2
Theory

This chapter describes the methods and techniques that were chosen to solve the task of crowd counting using LiDAR. An overview of existing solutions is provided, along with their advantages and disadvantages. Then we go through some important machine-learning concepts as well as the architecture of the deep-learning models we used in the implementation.

2.1 Existing Solutions

Previous research in this area primarily focuses on crowd counting using cameras. Camera technology has been around longer than LiDAR and has been more widely used in surveillance. There are three core approaches for crowd counting that have been explored: detection-based, regression-based, and density estimation [5, 6]. These three strategies will be discussed in the following subsections.

2.1.1 Detection-Based Crowd Counting

Object detection in images is a well-studied topic in computer vision. It has seen rapid development in the past 20 years, driven by advancements in machine learning. The development of object detection can be separated into two historical periods: before and after deep-learning [7].

Before 2012, object detection was done without the help of neural networks. The first algorithm to achieve real-time face detection was the Viola Jones Detector, created by P. Viola and M. Jones in 2001. It used sliding windows to search images for faces at different scales and positions. A few years later, N. Dalal and B. Triggs revealed the HOG (Histogram of Oriented Gradients) detector, mainly built for the purpose of pedestrian detection. The later stages of the 2000s were dominated by the Deformable Part-Based Model, proposed by P. Felzenszwalb in 2008. It was an extension of the HOG detector and used a divide-and-conquer strategy to detect even more classes than previously possible [7].

Convolutional Neural Networks (CNN), described in subsection 2.2.2, revolutionised the field of object detection in 2014, starting with the introduction of Regions with CNN (RCNN). It worked by selecting candidates for objects in an image and feeding them into a CNN to extract its features, which were then used as input to a linear classifier. From this, a large number of algorithms and improvements have been developed and are still being researched. Current strategies for object detection can be categorised into two types: one-stage methods and two-stage methods. One-stage methods achieve faster inference speeds, while two-stage methods generally achieve higher accuracy. Recently, research regarding the use of transformers in computer vision has led to further advancements in performance. The current state-of-the art model, Co-DETR, uses this technique [8] [7].

Detecting people in an image and counting them may seem like a natural approach to the task of crowd counting. However, crowded scenes are problematic for detectors. As partial occlusion and long distances are introduced, the accuracy of detection-based methods rapidly decreases. Figure 2.1 shows how a detection model fails to detect a partially occluded person. A method of detection-based counting has been proposed by B. Wu and R. Nevatia in 2005, building on the Viola Jones Detector [9]. Their method worked by detecting the edges of people and combining them. This technique allows the algorithm to detect partially occluded people better than traditional detection, however, it is still limited by low resolution and large distances [10].

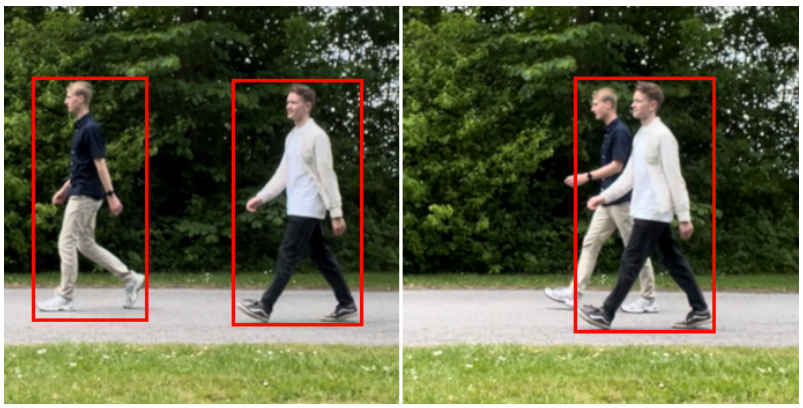


Figure 2.1: Two people are detected as one, showing a flaw in detection-based counting.

2.1.2 Regression-Based Crowd Counting

Regression is the process of learning the relationship between input and output data from an unknown function [11]. Machine-learning models can learn this relationship by processing examples with known outputs and updating their internal estimation of the function. There are multiple forms of regression: linear regression, Gaussian process regression, Bayesian regression, and many more, all suitable for different tasks. The most common type is linear regression, which aims to estimate the line that best represents the relationship between input and output. Typically, the mean squared error is used as a metric to determine how well the model performs.

An approach for crowd counting based on Bayesian regression has been proposed by Antoni B. Chan and Nuno Vasconcelos [12]. One of the main advantages of using Bayesian

regression is that it can utilise prior knowledge and is flexible, meaning it can handle both complex and linear models [13]. This regression-based model shows better performance compared to the detection-based model previously discussed. However, regression-based counting techniques lack spatial information [14]. Only the estimated size of the crowd is returned by the model, with no way of knowing the distribution of the individual people.

2.1.3 Density Estimation

Density estimation is a more recent approach to crowd counting [14]. The idea is that a machine-learning model takes an image of a crowd as input and outputs a new image, where every pixel represents how many people there are in that section of the input image. It results in a density map showing how the people in the crowd are distributed, as well as a way of telling where the crowd is the most dense, see Figure 2.2. The density map also has the important feature that the integral over all pixels results in the number of people visible in the image.

A density estimation model using a two-columned CNN was proposed by V. Ranjan et al. [15] in 2018. It works by producing both a low-resolution and a high-resolution density map and combining the results. This way, they can extract features on a larger scale with the low-resolution branch and use that to improve the high-resolution density map. At the time of publishing, their method reached state-of-the-art performances on popular datasets for crowd counting.

One issue that density estimation is susceptible to is vegetation and other messy backgrounds. Objects such as trees and bushes may be detected as highly dense areas of the image, resulting in counting mistakes in those sections, as can be seen in Figure 2.2. Z. Lui et al. tackle this by incorporating foreground segmentation and global scale information [6]. The foreground segmentation helps with eliminating the intricate background, while the global scale information adjusts the predicted count according to an overall density level in the image.

Another drawback of using density estimation is the tedious process of annotating the data. For regression-based models, the ground truth is just the number of people visible in the image. To accurately train a CNN to output density maps, the ground truth also has to be density maps. This means that the annotation process is to manually create these maps for the dataset. Usually, this means placing a pixel on all heads in the image and using a smoothing algorithm to blur that pixel over a larger area, as can be seen in Figure 2.2 (b). Annotating images with hundreds or even thousands of people is itself a large task, let alone annotating an entire dataset. However, while it may be difficult to find data for specific scenarios, there exist annotated datasets for crowd counting using density estimation [16].

2.1.4 Clustering in Point Clouds

Clustering is an important technique that can be applied to point clouds. It is the process of grouping together a selection of points based on a specific criterion [18]. This grouping can be done based on features such as distance to a point's neighbour, the reflectance of points, as well as other features depending on the task. Clustering can be used as a way to abstract a point cloud, meaning that instead of working with tens of thousands of points, one can work with only a handful of clusters. The technique allows for easier analytics on large point clouds, i.e. it is easier to track a cluster than to track individual points.

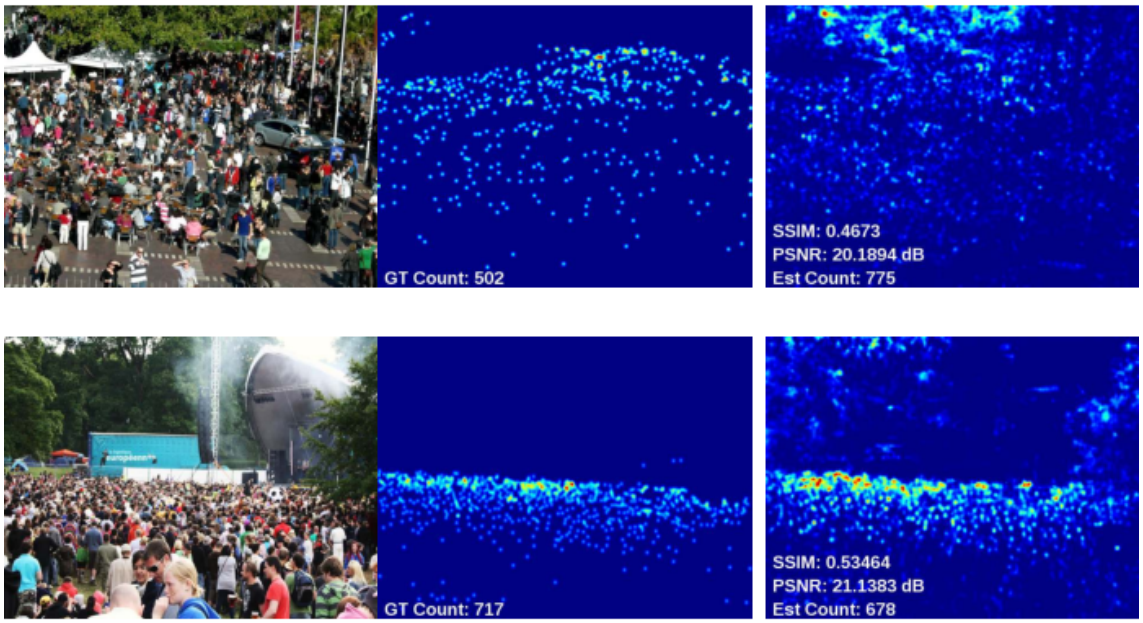


Figure 2.2: Intricate background misidentified as people. (a) Input image, (b) Ground-truth density map, (c) Estimated density map. Image from the paper A survey of recent advances in CNN-based single image crowd counting and density estimation [17]. Used with permission from the author.

A common clustering algorithm used when processing point clouds is Density-Based Spatial Clustering of Applications with Noise (DBSCAN). A great advantage of this algorithm compared to others, such as K-Means, is that the number of clusters to find is not specified before running the algorithm. Instead, two parameters—the maximum radius `eps` and the minimum number of neighbours `min_points`—are set, and DBSCAN will find all the clusters that fit these values. The algorithm works by finding core cluster points by checking if the distance to any of their neighbours is less than `eps`. If a point has more than `min_points` neighbours in this radius, these points are marked as a cluster. Clusters containing some of the same points are then merged until no clusters overlap. The pseudo code can be found in Algorithm 2.1, taking in the data $\boldsymbol{D}$, `eps`, and `min_points`. In Figure 2.3, it can be seen how `eps` affects the clustering. The number of clusters found directly depends on these parameters, which makes it important to tune them right in order to only find the clusters that are relevant. As a result of narrowing the parameters, more points are being labelled as noise.

The main problem with using clustering as a method for counting people in crowds is that as the crowd gets denser, the clusters tend to blend together. When two people get too close together, it becomes difficult for the clustering algorithm to differentiate the clusters, and they are counted as one person.

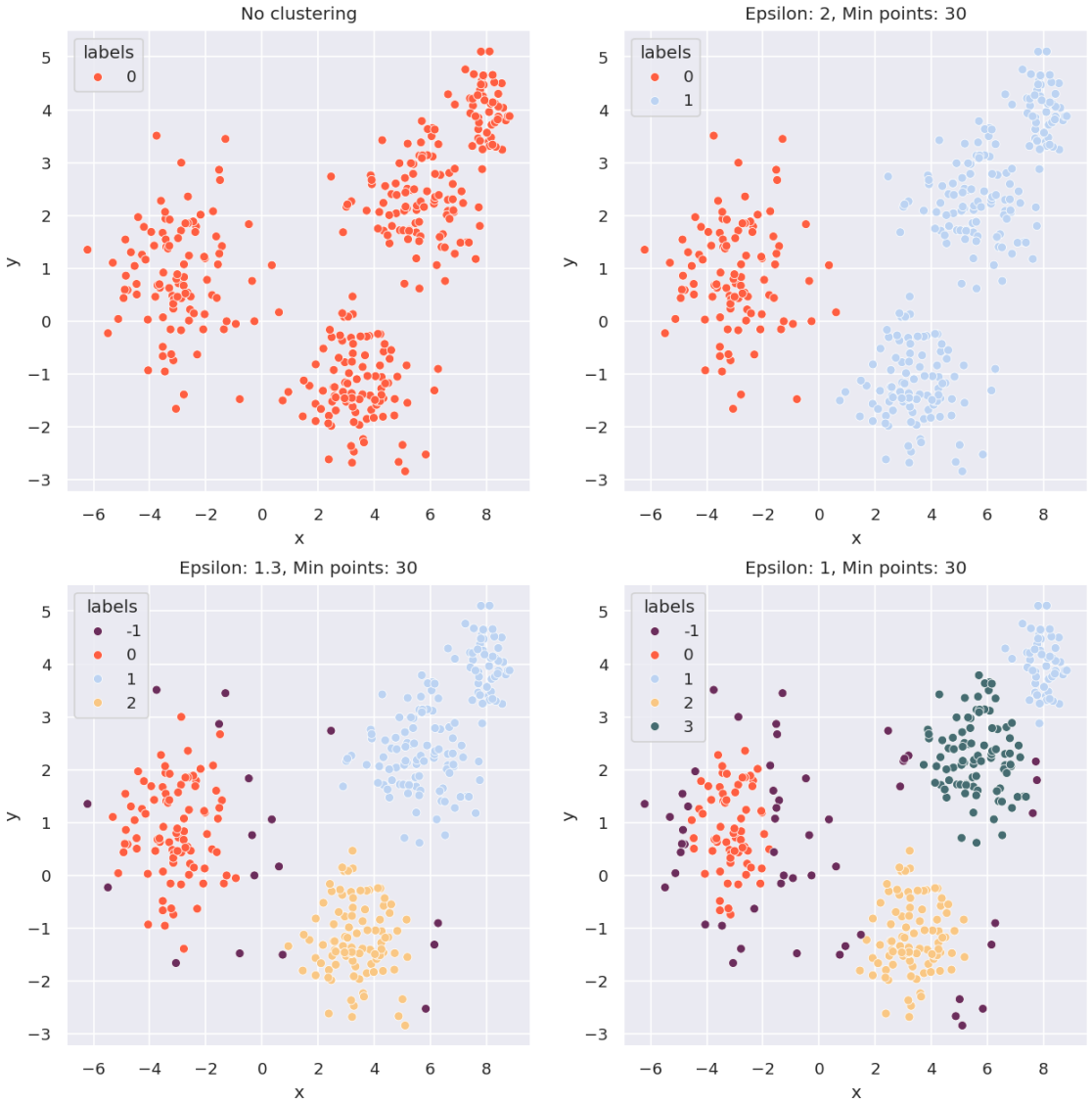


Figure 2.3: Demonstrating DBSCAN clustering on 2D data with varying `eps`. The labels indicate what cluster a point belongs to. Points with the label -1 are seen as noise points and do not belong to any cluster.

2.2 Machine-Learning Concepts

In this section, some important concepts used in machine learning are described.

2.2.1 Convolutions

In the context of machine learning, a convolution is a method used to filter out specific features from an input vector, e.g. an image [19]. It works by sliding a kernel (a small matrix of weights) over the image and calculating the scalar product for each position of the kernel. The output image will be of the same size or smaller than the input, depending on whether

Algorithm 2.1 DBSCAN

```
function DBSCAN(D, eps, min_points)
    C ← 0
    for all unvisited points p ∈ D do
        mark p as visited
        N ← regionQuery(p, eps)
        if len(N) < min_points then
            mark p as NOISE
        else
            C ← C + 1
            expandCluster(p, N, C, eps, min_points)
        end if
    end for
end function

function expandCluster(p, N, C, eps, min_points)
    mark p as part of cluster C
    for all p′ ∈ N do
        if p′ is not visited then
            mark p′ as visited
            N′ ← regionQuery(p′, eps)
            if len(N′) ≥ min_points then
                N ← N ∪ N′
            end if
        end if
        if p′ not part of any cluster then
            mark p′ as part of cluster C
        end if
    end for
end function

function regionQuery(p, eps)
    return all points within eps-radius of p (including p)
end function
```

zero-padding was used or not. If zero-padding is used, a frame of zeros is added to the input. Another hyperparameter is stride, which determines how far the kernel should be moved with each step. A 2D convolution, visualised in Figure 2.4, can be described as follows:

$$B(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega(i, j) A(x - i, y - j) \tag{2.1}$$

where A is the input matrix, ω is the kernel, and B is the result of the convolution. Every element of the kernel is included in $-a \leq i \leq a$ and $-b \leq j \leq b$.

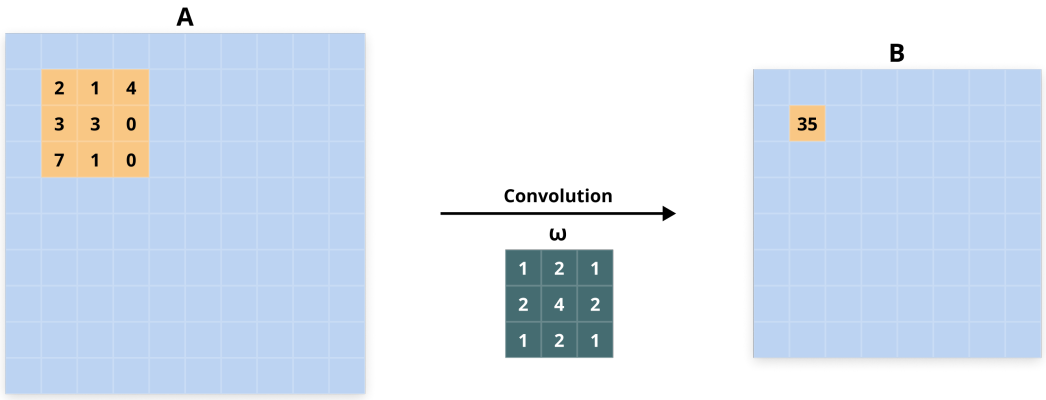


Figure 2.4: Illustration of how 2D convolutions work. Matrix A is convolved with kernel ω to produce matrix B. No zero-padding was used in this example, resulting in a smaller output matrix.

2.2.2 Convolutional Neural Network

A CNN is a neural network that takes advantage of convolution [19]. A CNN is usually constructed with multiple convolutional layers followed by a flattening process that ends with one or more fully connected layers. A convolutional layer requires fewer weights than fully connected layers and is more common when working with images [20]. To demonstrate the difference, suppose an input image of size 64x64 with three colour channels and a kernel of size 6x6x3. A convolutional layer would have 108 weights, while a single neuron would have 12,288 weights each in a fully connected layer. Furthermore, a fully connected layer is generally made up of more than one neuron, which means there would be linearly 12,288 additional weights per neuron. A common type of intermediate layer in a CNN is max pooling. This layer reduces the size of the input by selecting the maximum value within each kernel's area as the output.

2.2.3 Segmentation

Segmentation is a process for dividing something into different parts. In image segmentation, it could mean finding regions of specific objects in the image. When doing segmentation with machine learning, the goal is to predict a label for each pixel, meaning that all pixels of a car get the class label `Car`, and all pixels of humans get class label `Human`. The same concept holds for segmentation in point clouds. But instead of predicting labels to pixels, each point in the point cloud gets a label.

There are three types of segmentation: semantic segmentation, instance segmentation, and panoptic segmentation [21]. The types of segmentations are visualised in Figure 2.5. Semantic segmentation is when the label describes the object in that pixel, e.g. `Car`, `Human`, and so on. Instance segmentation is a combination of semantic segmentation and object detection. The result of instance segmentation is, therefore, both what type of object it is and which specific instance of the object it is. Finally, panoptic segmentation is a combination of semantic and instance segmentation. It uses instance segmentation for objects that should

be counted, such as people or cars, and it uses semantic segmentation for objects that do not have to be counted, such as trees or roads.

Figure 2.5: Visualises the different types of segmentation [21]. Used with permission from the author.

2.3 Deep Learning on Point Clouds

In this section, we will discuss how to use deep learning on point clouds. Deep learning is a form of machine learning that employs deeper network architectures, such as multi-layered neural networks. A point cloud is a set of points in $\mathbb{R}^3$ space, where each point has a x, y, and z value. Compared with images, point clouds are very irregular since there is no order between points and the number of points varies.

There are methods available to transform a point cloud into a more regular shape, such as the image grid transform and voxel transform. An image grid consists of a collection of images from different viewpoints generated from a point cloud, whereas voxels are regular grid cubes in 3D space. A voxel is considered occupied if at least one point from the point cloud is located within it. However, transforming the point cloud into voxels or image grids presents certain challenges. When using voxels, the size of the voxel must be determined. Very fine-grained voxels result in a large matrix, requiring some compression, while larger voxels create a smaller matrix but require greater compression. Additionally, these transformations can obscure natural invariances present in the original data.

Using one of these transforms would mean that CNN could be used as a network. However, to be able to use the point clouds themselves as input, the network has to be able to deal with the following three properties of point clouds:

- **Unordered**. The order of the points should not matter.
- **Interaction among points**. Points should be located in metric space and have meaningful neighbours.
- **Invariance under transformations**. Transformations such as rotating the point cloud, should not result in different segmentation and classification outputs.

2.3.1 PointNet Architecture

PointNet is a deep-learning architecture that is able to solve segmentation and classification tasks on point clouds. It was proposed by Qi et al. in 2017 [22] and was one of the first deep

network architectures that was able to use point clouds as input without having to transform them into image grids or 3D voxels in advance.

The architecture of PointNet is visualised in Figure 2.6. The main pipeline is for learning classification problems. It takes an input of `n` points and outputs a one-hot vector of size `k`, i.e. a vector where a single value is 1 and all other values are 0. Each value of the one-hot vector represents a class, and the index with value 1 is the corresponding predicted class for the input.

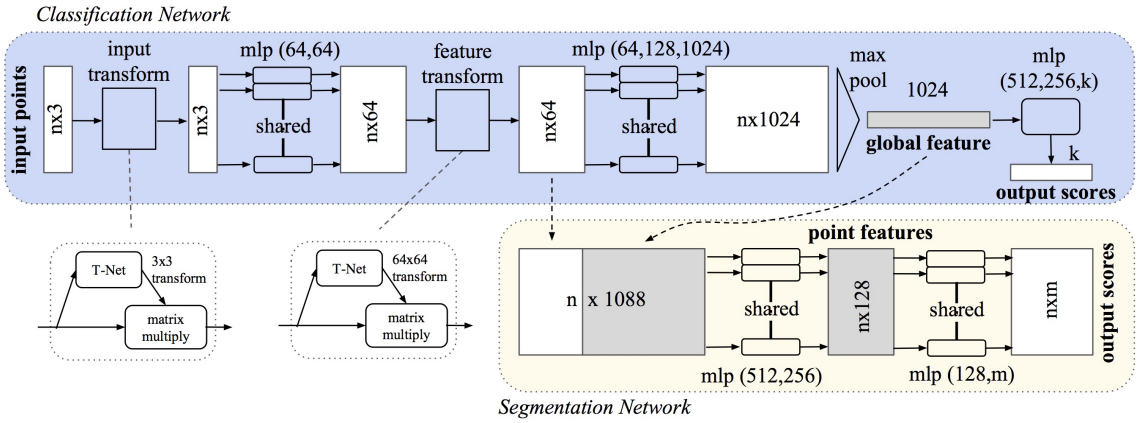


Figure 2.6: PointNet Architecture. The blue part is the classification network, and the yellow part is the segmentation network, which is an extension of the classification network. Image by Qi et al. [22]. Used with permission from the author.

The segmentation network is an extension of the classification network, with the difference that it returns a class for each input point. Here, `m` is the number of classes.

PointNet consists of three central modules: a symmetry function, local and global information aggregation, and a joint alignment network.

For the symmetry function, PointNet utilises max pooling. The purpose of a symmetry function is to ensure that the result is independent of the order of the inputs. The max pooling in PointNet has an input shape of `n` x 1024 and an output shape of 1024. The output is the largest value in each column. As seen in Figure 2.6, all points are treated separately before the max pooling, which ensures that the input order does not matter.

Local and global information aggregation ensures that points can interact with each other. By concatenating all point features with the global feature gained from max pooling, every point obtains information about the point cloud. This is visualised at the beginning of the segmentation network part of Figure 2.6.

The 3x3 and 64x64 T-net transforms in the architecture are two joint alignment networks. They make sure that the network is invariant to rigid transformations, i.e rotations, translations, and reflections.

2.3.2 PointNet++ Architecture

PointNet++ is a successor to PointNet, created by the same people as the original PointNet [23]. PointNet++ aims to solve two issues with its predecessor. The first issue involves

managing features with different scales in the same point cloud. And the second is to be able to process point clouds of varying densities.

In order to solve these problems, PointNet++ employs a hierarchical neural network architecture for processing point clouds. The hierarchical architecture enables the network to find both local and global features of the data. The PointNet++ architecture is visualised in Figure 2.7. PointNet++ uses sampling and grouping to find local regions of the point cloud, followed by a forward feed through an original PointNet network for each local region, using PointNet as described in section 2.3. The layers work as follows:

- **Sampling Layer**. This layer takes the whole point cloud as input and finds centroids for all local regions by doing iterative farthest point sampling (FPS). FPS begins by selecting a random point as the first centroid. The other centroids are found by iteratively selecting the point farthest away from the previously selected centroids. FPS gets better coverage of the point cloud compared with random sampling.
- **Grouping Layer**. With the centroids found in the sampling layer, this layer can create local regions. PointNet++ uses a ball query to find all points within a certain radius of a centroid with an upper limit of K points. Another method would be to use K nearest neighbour (kNN) search to find the K nearest points. Ball query is more favourable because it assures a fixed region scale.
- **PointNet Layer**. This layer extracts features from each local region by doing a forward pass through a PointNet network. However, before the local region is passed through the PointNet, it is translated to a local coordinate system with the centroid set as the origin. The same PointNet is used for each local region, similar to how the same kernel moves over an image in a convolutional layer.

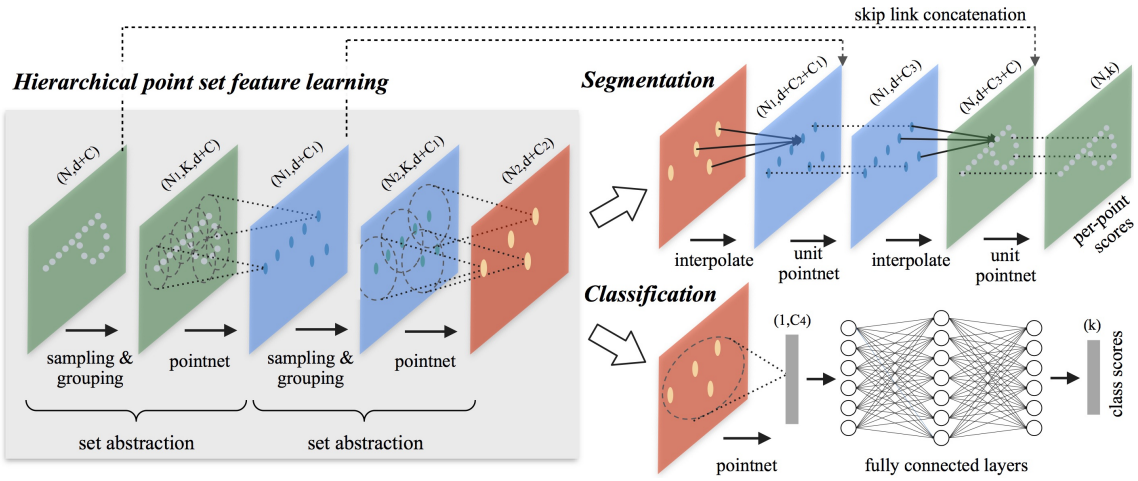


Figure 2.7: PointNet++ Architecture. Each step includes sampling, grouping, and a small PointNet. The final part of the network is divided into a segmentation network and a classification network. Image by Qi et al. [22]. Used with permission from the author.

As seen in Figure 2.7, these three layers are repeated until the set abstraction is completed. From there, it is possible to either do segmentation with point feature propagation or do

classification with a multi-layered perceptron (MLP). A MLP is a type of neural network where multiple layers of neurons are connected, and the data is fed forward through the network to produce an output at the last layer.

Since every point from the input point cloud needs a label in segmentation, the last abstracted point cloud has to be up-sampled to the original size. The segmentation process begins by interpolating features for points that were discarded during each grouping layer. The features $f(x)$ are the inverse distance weighted average of the features of the k nearest neighbours in the previous layer and are calculated with Equation 2.2. A skip link concatenation is applied, where the interpolated features are concatenated with the features from the corresponding set abstraction. Before repeating the interpolation process again, a "unit PointNet" is applied to each point. The "unit PointNet" acts like a one-by-one convolution in CNNs. When the point cloud is fully upsampled, a per-point score for the segmentation has been found.

$$f(x) = \frac{\sum_{i=1}^{k} \omega_i(x) f_i}{\sum_{i=1}^{k} \omega_i(x)} \quad \text{where} \quad \omega_i(x) = \frac{1}{d(x, x_i)^2} \tag{2.2}$$

2.4 Unreal Engine 5

The simulator used to generate the data for this thesis was built in UE5. UE5 is a 3D game engine built by Epic, especially used in game development, but it can also be used as a simulator [24]. Games built on UE5 include Fortnite and Ark: Survival Ascended. The Engine provides tools for creating realistic environments with accurate visuals, lighting, and physics while still running efficiently in real time on the computer. UE5 is mainly built for game developers who want to create worlds for their games. However, in this master's thesis, it was used to simulate and record crowds with a LiDAR sensor.

Chapter 3
Approach

This chapter begins by discussing how all of our data was generated, describing the different datasets used throughout the master's thesis.

In the sections 3.2, 3.3, and 3.4, we will describe how all of our approaches were implemented. The first approach, described in section 3.2, called human clustering, was based only on a clustering algorithm finding the clusters of all people. The approach described in section 3.3 is called head segmentation. This approach used a deep-learning model to segment heads from the point cloud, then a clustering algorithm was applied to count the number of heads. Finally, two density estimation approaches are described in section 3.4. Both of these were deep-learning approaches trying to estimate the number of people per point in the point cloud, meaning that the sum over all points estimates the number of people in the entire point cloud. A graphical overview of the entire workflow is visualised in Figure 3.1.

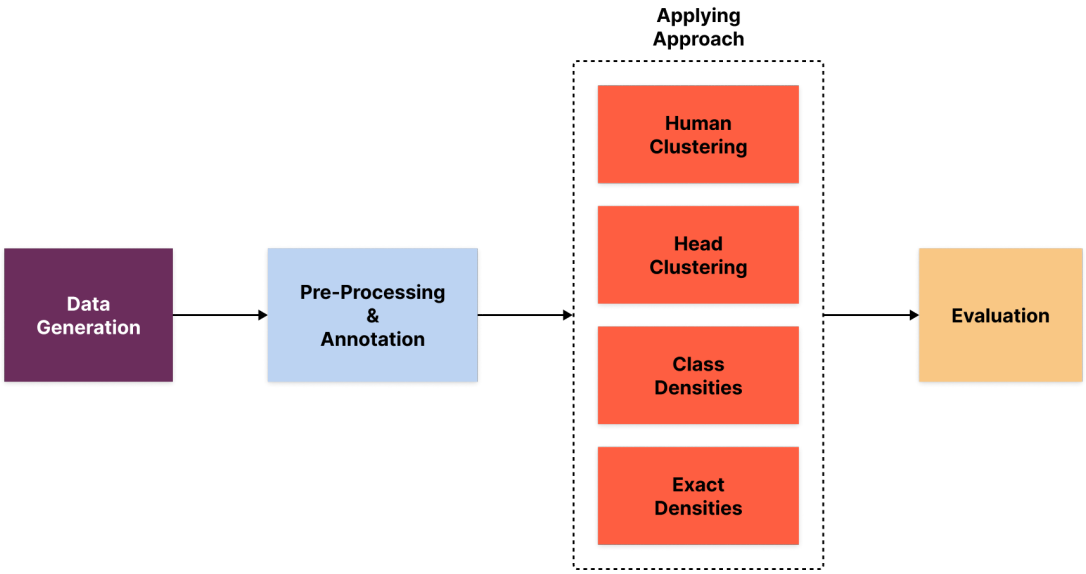


Figure 3.1: Graphical overview of the workflow.

3.1 Dataset

To solve the problem of not having any data to work with, we could either chose to record LiDAR data from the real world, or synthetically generate it using a simulator such as UE5. We opted for generating the data since it allowed for automatic annotation and faster data gathering. Some real data was collected as well, to be used for visual testing.

3.1.1 Simulated Dataset

As previously mentioned, UE5 was used to generate the simulated data. Another alternative would have been to use an already existing LiDAR simulation tool called Carla, which is mostly used to create simulated data for autonomous driving systems, built on Unreal Engine 4 (UE4) [25]. Although Carla has been used in master's theses prior to ours with somewhat successful results, as in "Pedestrian detection and tracking in 3D point cloud data on limited systems" [26] and "Segmentation, Classification and Tracking of objects in LiDAR Point Cloud Data Using Deep Learning" [27], we decided to use UE5. Our decision was based on the fact that we already had some experience using UE5, and we felt that we had more freedom in controlling how the LiDAR should behave compared to the Carla simulator. Additionally, UE5 is a well-established tool for creating simulations [28].

The Epic Games Marketplace offers assets, both paid and free, that can be used by creators in UE5, some of which come with pre-built worlds. The worlds we chose to work with were a park map called City Park and an auto-generative city map. Both of these were free to use. We also created our own map, where we could easily control the number of people and the size of the space they could walk around in. This map was only used to create controlled test scenarios, as described further in subsection 3.1.3. The city map had generative properties, meaning that the city was randomly generated from a palette of a few different buildings, randomly placed trees, and varying road layouts. To maximise entropy, we wanted the world to include variance in ground level, such as stairs or slopes, which the city map lacked. The park contained multiple ground levels, with staircases connecting them. While the park did not have generative properties, it was still a large enough world with multiple locations suitable for recording. Some interesting locations include a fountain park with staircases and a baseball field, which, when filled with people, could look like a concert crowd.

A crowd was simulated by manually placing pedestrian paths in the editor that humans, called MetaHumans, would follow. MetaHumans are a type of non-playable character available in the Epic Games Marketplace. They come with a set of intelligence traits that allow them to navigate the pedestrian paths fluently without clipping through each other or other objects in the scene. The paths that they followed were not strict, meaning that they did not have to follow exact paths but rather roam around freely close to the paths, resulting in crowds that appeared more lifelike.

The data was generated by placing LiDAR modules around the crowd. For each scene, we used between three and five LiDAR modules in order to capture more data from the same crowd. The LiDAR modules were placed in a way that made them have completely different views of the crowd by placing them at different locations with varying heights and angles. The reason we wanted more than one LiDAR in each scene was that we could generate more varied data faster this way.

One LiDAR data file contained all the points from a single point cloud. Each line in the file had information about one point, mainly: `x`, `y`, `z`, `distance`, `reflection intensity`, `line_index`, `object_ID`, and `annotation_ID`. The coordinates were represented by `x`, `y`, and `z`, measured in Unreal units. These units were interpreted as centimetres when using the datasets. The `distance` parameter represented the distance from the LiDAR sensor to the point in Unreal units. The `reflection_intensity` was a floating point number between 0 and 1, where 1 is total reflection while points with `reflection_intensity` close to zero are less reflective. The `Line_index` denoted which line the point belonged to. The `object_ID` was a unique number shared with points that came from the same MetaHuman. Lastly, `annotation_ID` was used differently depending on how we wanted to annotate the data for the different approaches, described in sections 3.3, 3.2, and 2.1.3.

When recording with the LiDAR sensor, we excluded most of the background points from the data. This simplified the training process, and by never including them directly, we did not have to send the data through a background subtraction step when training. However, since we wanted the models to be trained with data that resembles real data, where background subtraction may not always be perfect, we did not exclude all background points.

3.1.2 Real Data

Even though we did not opt for real-world LiDAR recordings, we have two hours of LiDAR data from a real crowd. The data was recorded at an event at Malmö Arena for the employees of the company we wrote for. This was a great opportunity for us to get a chance to work with a real LiDAR sensor while at the same time being able to collect some valuable data.

The LiDAR was placed 2.40 metres above the floor in the perimeter corridor of the arena. The LiDAR overlooked four entrances into the arena. We chose this location because a lot of people would gather around the entrances in the beginning and during the pauses of the event. Before the event began, the space in front of the LiDAR was nearly empty of people, and as the entry to the arena began, more and more people occupied the space. This increase in people density meant that we got data on crowds of varying sizes and densities, ranging from a couple of people up to a blue zone crowd, as described in section 1.5.

As mentioned in subsection 3.1.1, it is time-consuming to annotate LiDAR data manually. Therefore, we decided not to annotate any of the real data we captured at Malmö Arena and thus not use it for training. Instead, we only used it for visual evaluation of our approaches.

3.1.3 Evaluation Datasets

To evaluate the performance of our approaches, we produced three different datasets, aiming to test the robustness of our approaches. The tests were designed to evaluate the performance with respect to people quantity, crowd density, and distance. The three datasets had 1,000 samples each.

For the evaluation with respect to people quantity, a dataset containing samples from ten scenes was used. Every scene had a different number of people, ranging from 1 to 750. From this evaluation, we wanted to find out in what ranges our approaches perform better and in what ranges they perform worse.

In the dataset for evaluating crowd density performance, every sample had 100 people in it, while the area they were placed in varied from 100 square metres up to 784 square meters.

In other words, the smallest square the crowd was bound to was 10 by 10 metres, and the largest was 28 by 28 metres. An increase of two metres in side length was applied in order to get ten different area sizes in the test. The information we wanted to gather from this test was how well our approaches could handle dense and sparse crowds.

For the distance evaluation test, a dataset with varying distances from the LiDAR sensor to the crowd was made. Just like the density test, every sample had 100 people in it, but in this case the area was constant at 400 square metres. The closest distance was 19 metres, and the furthest distance was 100 meters. Again, ten different scenes were made, with the distance increasing by nine metres per scene. We wanted this test in order to evaluate if there were any differences in the result depending on the distance to the crowd.

3.2 Human Clustering Approach

The first and most simple approach to crowd counting was to find all clusters in a point cloud and count them. Since people in a point cloud appear as clusters, an algorithm such as the one described in subsection 2.1.4 could be used to find them. The last step is to simply count the number of clusters found by the algorithm. Therefore, this approach uses no deep learning, only the clustering algorithm. Before any of these steps could be applied, the point cloud had to be filtered to remove any background points that might interfere with the clustering. All of these steps form a pipeline, outputting a number for a given point cloud. The pipeline can be seen in Figure 3.2, and the individual steps are further described in the following subsections.

Figure 3.2: Pipeline for the human clustering approach.

3.2.1 Background Subtraction

To improve the performance of the clustering algorithm, all irrelevant points in the point cloud should be filtered out, leaving only the points composing humans. This could be achieved using a background subtraction step. This module works by taking in a point cloud that serves as a background model and comparing it to another point cloud. Ideally, this background model should contain no objects except for the ones that should be filtered out, and the background should be as similar as possible to the rest of the data. The subtraction works by voxelising the background model and saving the voxels (3D version of a pixel) containing points in a set. Voxelising is the process of dividing 3D space into a fixed-size grid of voxels. The next step is to simply remove all points in the input point cloud that lie inside one of the background voxels. The accuracy of the filtration depends on both the background model and the voxel size. A larger voxel size accepts more error in the background model while potentially removing more points from the input than necessary.

The pseudocode outlining the background subtraction algorithm as described above can be found in Algorithm 3.1. The algorithm takes in three parameters: $\boldsymbol{B}$: the point cloud of

the background model; X: the point cloud that is to be background filtered; and `size`: the size of the voxels.

The same background subtraction algorithm was used in all following approaches.

Algorithm 3.1 Background subtraction

```
1:  function backgroundSubtraction(B, X, size)
2:      Y ← []                                          ▷ Output points
3:      V ← []                                          ▷ Background voxels
4:      for all b ∈ B do                                ▷ Voxelise all points in background
5:          v ← voxelCoordinates(b, size)
6:          Add v to V
7:      end for
8:      for all x ∈ X do                                ▷ Filter the input
9:          v ← voxelCoordinates(x, size)
10:         if v ∉ V then
11:             add v to Y
12:         end if
13:     end for
14:     return Y
15: end function

16: function voxelCoordinates(p, size)
17:     y ← round(p ÷ size) × size
18:     return y
19: end function
```

3.2.2 Clustering

The clustering part of this pipeline was relatively simple. It used an existing implementation of the DBSCAN clustering algorithm by scikit-learn, similar to Algorithm 2.1. The process of finding the parameters `eps` and `min_points` consisted of performing tests on a training dataset with different combinations of the parameters. Using the results from these tests, an optimal set of parameters could be chosen.

Finally, to get the total number of people in the point cloud, all of the clusters found could be counted.

3.3 Head Clustering Approach

This approach to solving the task includes two main parts. The first part was to find all the heads of the people by using a segmentation deep-learning model, and the second was to cluster the heads. It was reasonable to assume that heads are usually further separated than the bodies of people in crowded situations, making them easier to cluster. Depending on the density of the crowd, people may make contact shoulder-to-shoulder while their heads stay separated. To take advantage of this, a model has been trained to segment the heads,

meaning it can filter out all the points that do not belong to people's heads. This process is further described in the following subsection. With the point cloud only containing heads, the same clustering technique as in section 3.2 could be applied. These steps are visualised in the pipeline in Figure 3.3.

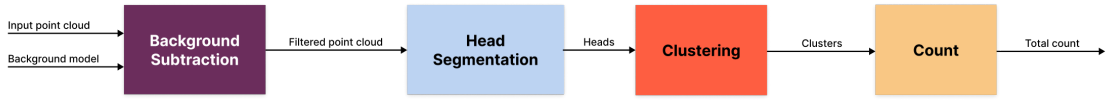


Figure 3.3: Pipeline for the head clustering approach.

3.3.1 Segmentation

The segmentation model was built in Python using the PointNet++ architecture, as described in subsection 2.3.2. The dataset contained crowds in the range of 1–700 people, with varying densities of the crowd as well as varying distances to the LiDAR sensor. The background was mostly filtered out when generating the data, leaving a small amount of noise to help with the generalisation of the model. This also meant that we could skip the background subtraction step in our pipeline when training.

When studying the generated data closer, it was noticed that the class distribution was not even. In our data, we have two classes: `other` and `head`, where the `other` points massively outnumbered the `head` points. This is problematic when training the machine-learning model since it produces bias in the model and inaccurate metrics. The training would indicate unreasonably high accuracy at the start of training, predicting the `other` class for every point in the point cloud. To help the model make sense of this unbalanced data, we incorporated class weights into the loss function—cross entropy in this case. These class weights adjust the impact a training example has on the model's weights. In our case, it would mean that an error on the `head` class would have a larger impact than an error on the `other` class. Adding this, as well as implementing more relevant metrics such as precision, recall, and f1-score, helped us improve the segmentation model. Figure 3.4 shows an example of how an annotated point cloud may look.

The PointNet++ architecture requires the input to always have the same shape, meaning the point cloud has to have the same number of points for every sample. This requirement imposed a few challenges. It is not reasonable to assume that a point cloud from a LiDAR will have the same number of points for every frame, since reflections and noise will cause points to appear and disappear unpredictably. Furthermore, when filtering out the background, the number of points remaining depends entirely on the number of moving objects in the scene, which in our case is the crowd size. Therefore, we had to downsample or upsample the point cloud to a fixed number of points before our model could process it. This was done using a fixed-point transform that either removed or duplicated points uniformly until the desired number of points was met. To minimise data loss in this step, we found the median value for the number of points per sample in our train dataset and applied the fixed point transform to that number. This means that there are as many upsamples as downsamples in total.

In addition to the fixed-point transform, normalisation transforms are also applied. Normalisation is the process of bringing all data to a similar scale, between 0 and 1. This will

help the model become scale-invariant and generally improve its performance. We start by centering the point cloud around the origin (0, 0, 0) by subtracting the mean from all points:

$$p_i' = p_i - \frac{1}{N}\sum_{j}^{N} p_j \tag{3.1}$$

All of the point coordinates are then divided by the largest coordinate in the sample, resulting in a normalised point cloud:

$$p_i'' = \frac{p_i'}{max(abs(P))} \tag{3.2}$$

When utilising the trained model for inferring new, unseen data, the sample is passed through the network multiple times. Since we apply the previously mentioned transforms, the output will vary when predicting the same data. Predicting multiple times allows us to create a statistical probability for every point to decide what class it should have, depending on the number of times it was predicted to be a certain class.

These transforms also had an impact on the clustering part of the pipeline. If we were to run the clustering algorithm on the data after it had been normalised, the distances between points would vary depending on how much the point cloud was scaled during the transforms. This would mean that the clustering parameters, mainly `eps`, had to be different for all samples. To solve this, we store the scaling factor and offset that were used in the transformations and apply them in reverse to get the original point cloud back to the way it was before clustering. This ensures the clustering is always applied to point clouds with the same units of measurement.

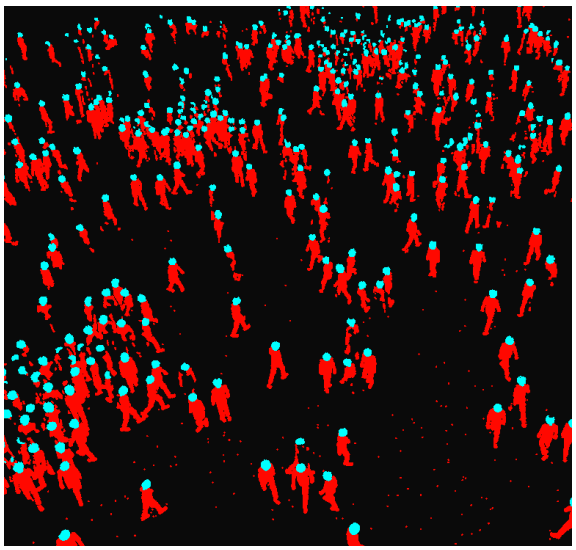


Figure 3.4: Annotated point cloud from the training dataset. Blue points are `head` class, red points are `other` class.

3.4 Density Estimation Approaches

As mentioned in subsection 2.1.3, density estimation has reached state-of-the-art performance in counting people in images. Therefore, it seemed interesting to attempt to replicate this technique on point clouds. A density map could be represented in a few different ways in 3D data, and it is not clear which would be the optimal representation. In the following subsections, two approaches using density estimation are described, both based on deep learning.

3.4.1 Class Densities

Density estimation and segmentation are closely related in the sense that every data point is assigned a value. Using this logic, a method for assigning classes that represent densities to the points in the point cloud was formulated. To decide what class a point should be assigned in the annotation process, we look at the total number of points that compose a person in the point cloud. All of these points will share the same class k_p, which can be found using the following formula:

$$k_p = round\left(\frac{K}{1 + e^{-a(n_p - C)}}\right) \tag{3.3}$$

where K is the number of classes, a is the area covered, n_p is the number of points that compose person p, and C is the centre of the function. Figure 3.5 (a) displays an example of how this function may look. When analysing our generated data, we observed that the distribution of points comprising an individual was not uniform; rather, it resembled a normal distribution, as can be seen in Figure 3.5 (b). Because of this, we decided to use this sigmoid-like function for class assignment, as that would even out the number of samples per class, therefore creating a more balanced dataset.

After the model had predicted classes for each point in a point cloud, we could use the following equations to find the density, d_p, for a point:

$$n_p = min\left(\frac{-ln\left(\frac{K}{k} - 1\right)}{a} + C, 2C\right) \tag{3.4}$$

$$d_p = \frac{1}{n_p} \tag{3.5}$$

Using these equations, the sum of all d_p belonging to person p will be close to one. The reason it is not exactly one is because of the rounding in Equation 3.3. This also means that the larger K is, the more accurate the densities will be, which is not suitable for machine learning in general since adding more classes makes it more difficult for the model to differentiate them.

The last step in the pipeline, as visualised in Figure 3.6, is to sum up all the densities for each point to get the total count of people.

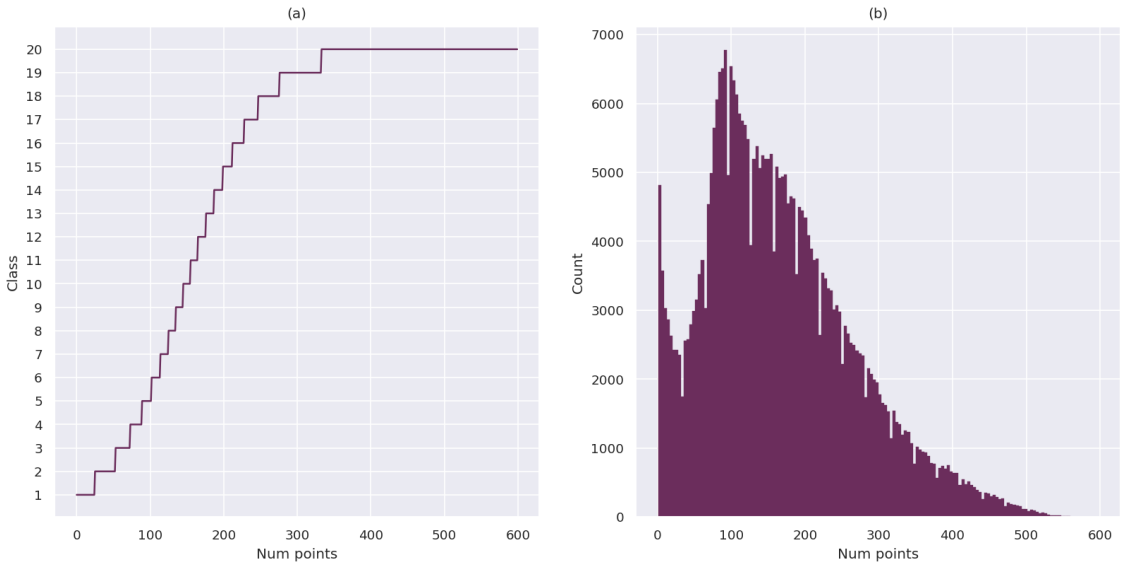


Figure 3.5: (a) Graph of Equation 3.3, using $K = 20$, $a = 0.02$, $C = 150$, and varying n_p. (b) Graph displaying the distribution of the number of points per person throughout the dataset.

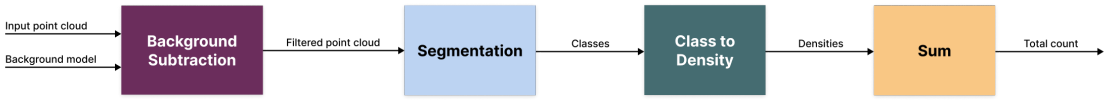


Figure 3.6: Pipeline for the density classes approach.

To understand this better, an example will be walked through. Assume a scene containing two people at different distances from the LiDAR. In the resulting point cloud, the two people will be composed of some number of points, say 195 points for the first person and 113 points for the second. Assuming perfect background subtraction, these are the only points that remain. Using Equation 3.3 and the same parameter values as in Figure 3.5 (a), we find that all of the points composing the first person should be assigned class 14, and all points composing person 2 should be assigned class 6. These are the classes that the segmentation model will attempt to predict. To turn these classes into densities that can be used to find the number of people in the scene, Equations 3.4 and 3.5 have to be used. Using these equations together with the predicted classes, all of the points belonging to the first person get the density 0.005198, and the second person's points get the density 0.009291. Summing up all the densities in the point cloud should then give an estimate of the number of people in the crowd. Since we assumed perfect background subtraction, this means: $195 * 0.005198 + 113 * 0.009291 = 2.063493 \approx 2$.

3.4.2 Exact Densities

To get around the rounding problem with using classes for densities, this approach modifies the network to output a floating point value for each point in the point cloud instead of a class. The pipeline for this approach is shown in Figure 3.7. The main modification to the

network was a change in the loss function. As mentioned in subsection 3.3.1, cross entropy has been used for the other approaches, which should only be used for classification and segmentation problems. Therefore, the loss function was changed to mean squared error instead.

The annotation process was simpler compared to the one used for class densities. The annotated data consisted of densities calculated using Equation 3.5.

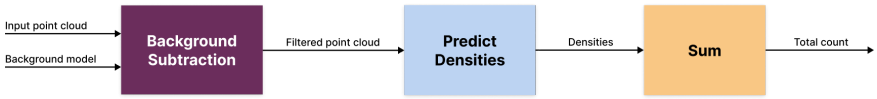


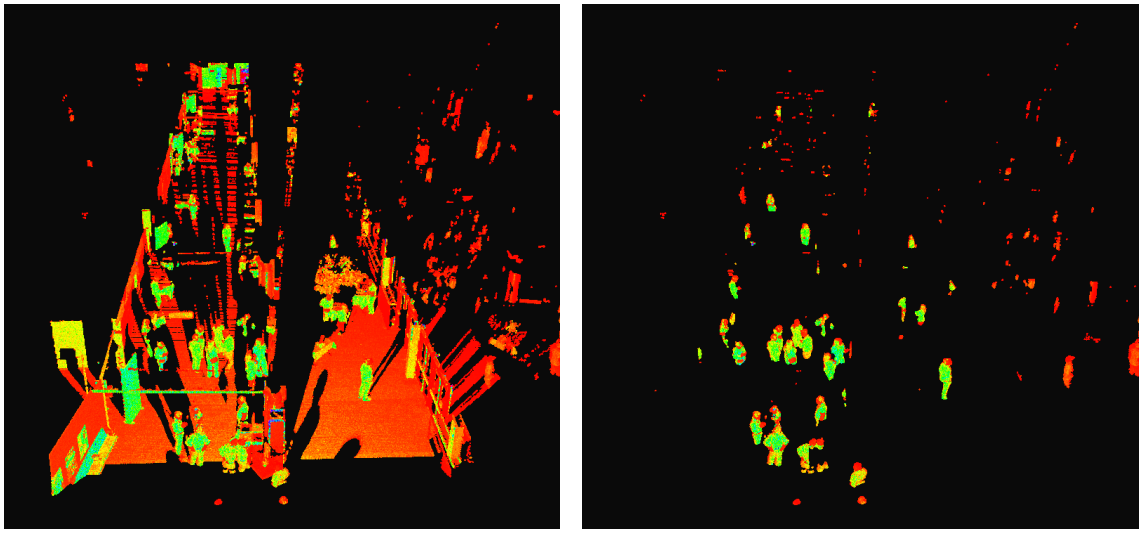
Figure 3.7: Pipeline for the exact densities approach.

Chapter 4
Results

This chapter will begin by showing the results for background subtraction and will then continue by presenting all the results we got from each approach we have implemented. Every approach will include results for the three tests: quantity, density, and distance, as well as some visualised results for real-world samples.

4.1 Background Subtraction

With a voxel size of 0.25 metres, a background subtraction going from Figure 4.1 (a) to (b) was possible. We found that a bigger voxel size removed too much from the foreground, such as lower parts of legs or people close to walls. On the other hand, if the voxel size was smaller, fewer background points were removed.



(a) Before. **(b)** After.

Figure 4.1: Captured from the perimeter corridor of Malmö Arena.

4.2 Human Clustering Approach

In this section, the results from the human clustering approach, as described in section 3.2, are presented.

4.2.1 Finding Clustering Parameters

The result for finding the DBSCAN parameters `eps` and `min_points` is shown in Figure 4.2. We varied `eps` between 10.0 and 40.0 and `min_points` from 4.0 to 32.0. Table 4.1 shows that `eps` = 30.0 and `min_points` = 8.0 had the lowest mean absolute error of 24.41. Therefore, we decided to use these parameter values when predicting on the evaluation datasets.

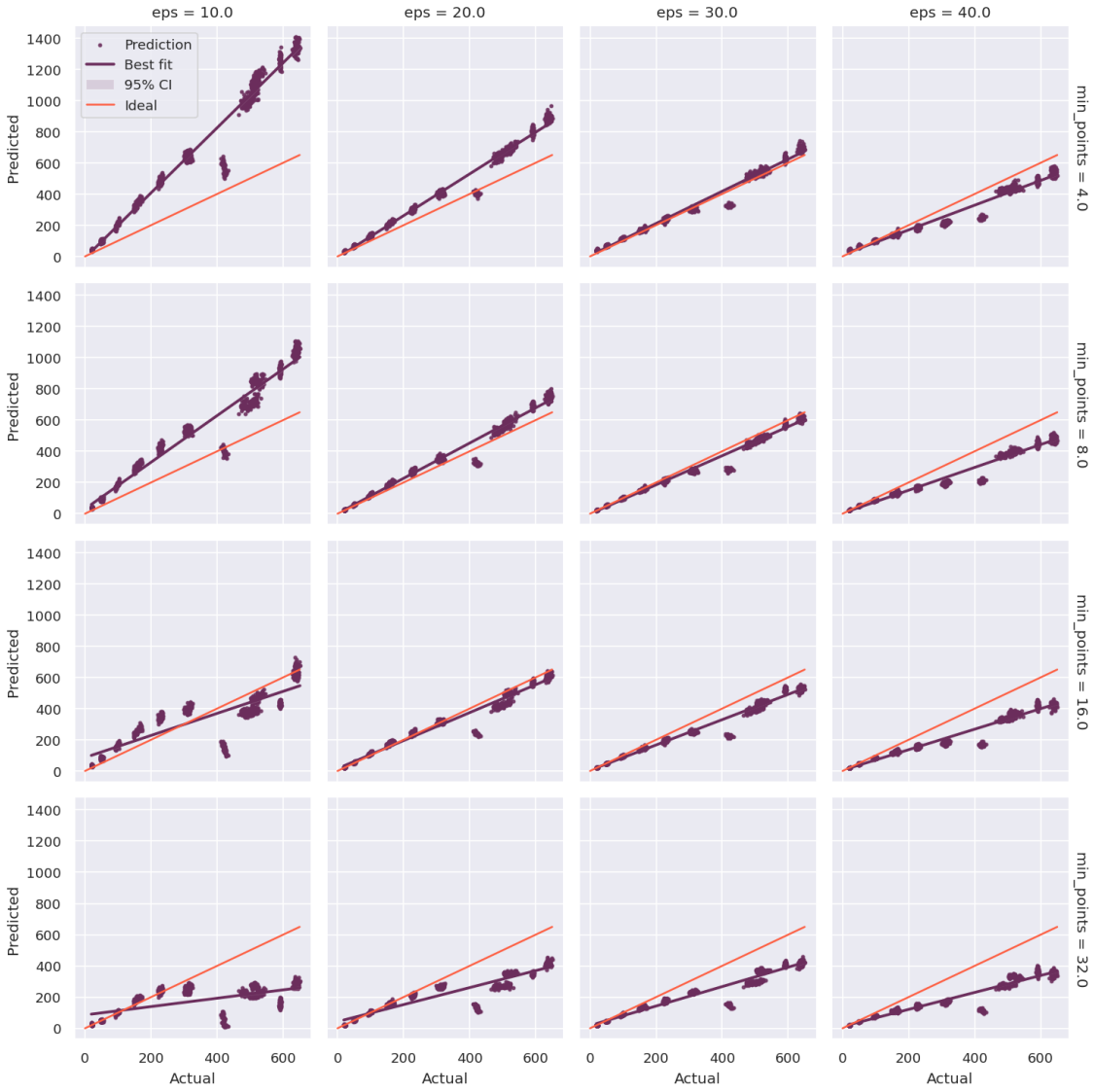


Figure 4.2: Regression estimates for all combinations of `eps` and `min_points`.

Table 4.1: Mean absolute error for all combinations of `eps` and `min_points`.

	eps = 10.0	**eps = 20.0**	**eps = 30.0**	**eps = 40.0**
min_points = 4.0	332.89	101.78	25.61	61.46
min_points = 8.0	190.34	50.78	24.41	81.89
min_points = 16.0	86.86	28.60	55.65	103.46
min_points = 32.0	156.24	102.97	101.56	132.42

4.2.2 Evaluation

Running the evaluation datasets through the human clustering pipeline resulted in the following graphs.

In Figure 4.3, the predictions for the quantity test follow the ideal line closely. The performance goes down when the number of people in the crowd increases, however, it still performs well at the upper limits of the test. The mean absolute error was 10.99 with respect to all test samples. However, as seen in Figure 4.4, the mean absolute error was 5.6 times lower at 1.95 when there were less than 100 people in the scene.

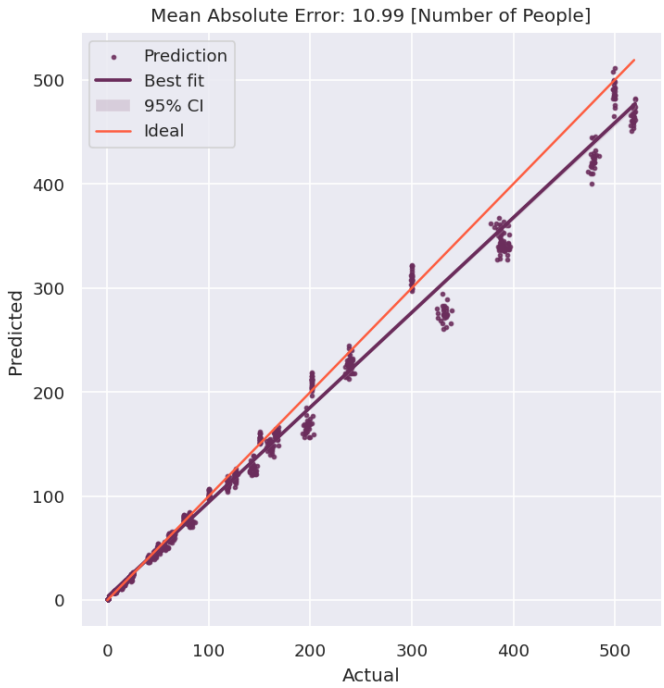


Figure 4.3: Quantity test results from the human clustering approach.

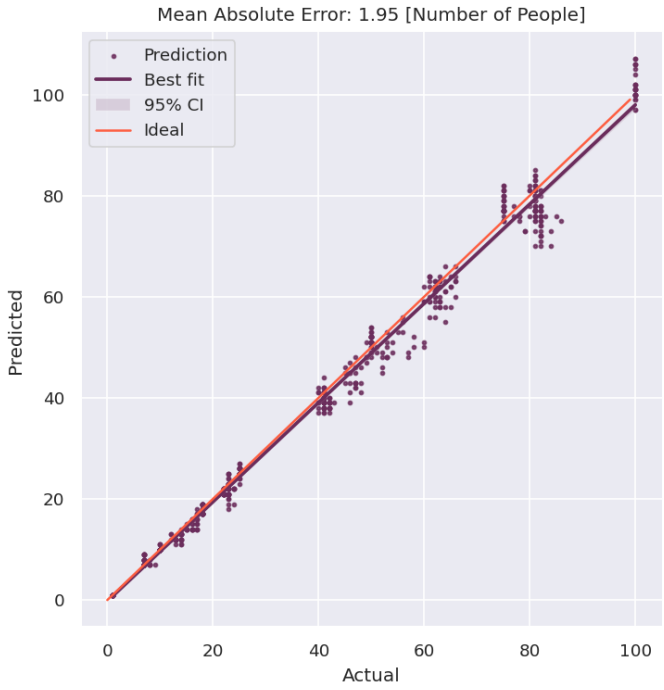


Figure 4.4: Quantity test results from the human clustering approach for crowds with less than 100 people.

Figure 4.5 shows the result for the density test. The median is close to error zero regardless of the area, where areas of 256, 324, and 484 square metres are the closest to error zero. The median is always positive, which means that the clustering algorithm finds more clusters than it should have in the majority of the samples. Figure 4.5 does also show that the distance between the lower and upper bounds of the whiskers is larger for denser crowds compared to sparse crowds.

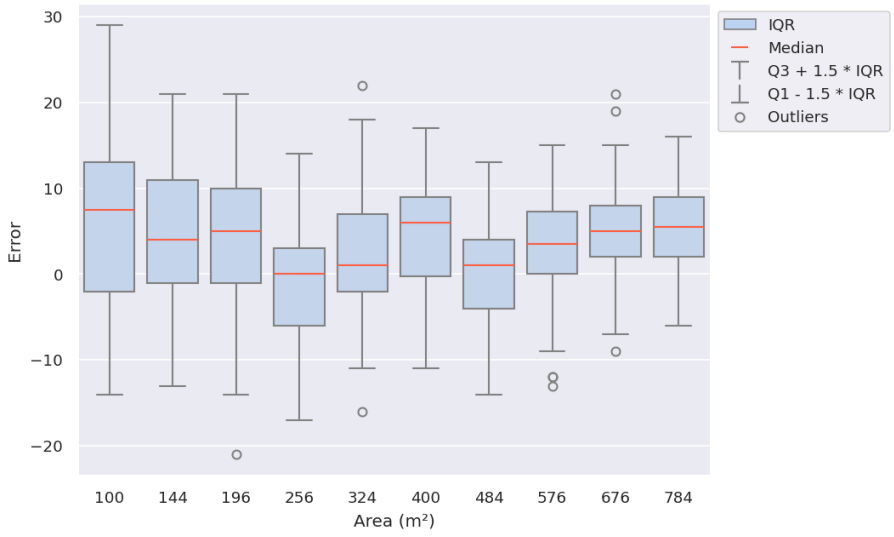


Figure 4.5: Density test results from the human clustering approach.

Continuing to the distance test, Figure 4.6 shows that the error was somewhat consistent between the distances of 28 and 64 metres, except for the error at 37 metres. When the distance exceeded 73 metres, the median error went from -5 to -15, which indicated that the error was getting worse when the distance increased.

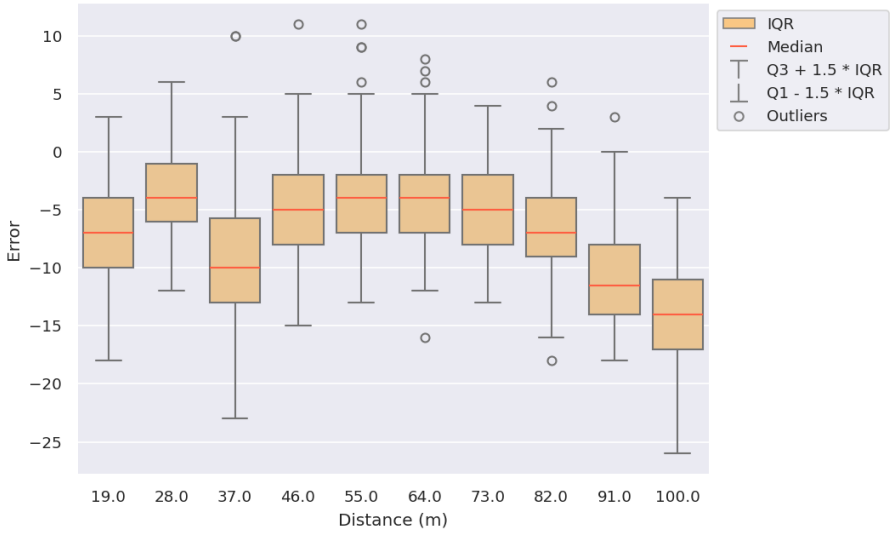


Figure 4.6: Distance test results from the human clustering approach.

When the human clustering pipeline was used with real data, it had problems separating people standing close together and filtering out noise. Figure 4.7 (left) shows the clustering results from a point cloud taken at the Malmö event, discussed in subsection 3.1.2. The coloured dots represent the centroid of each cluster. At the bottom left corner of the image, the model has failed to separate four people and counted them as one. It also found multiple clusters at the top that should have been counted as noise.

Figure 4.7 (right) is an example of the clustering results from simulated data. The clustering algorithm found 63 people when the actual count was 55.

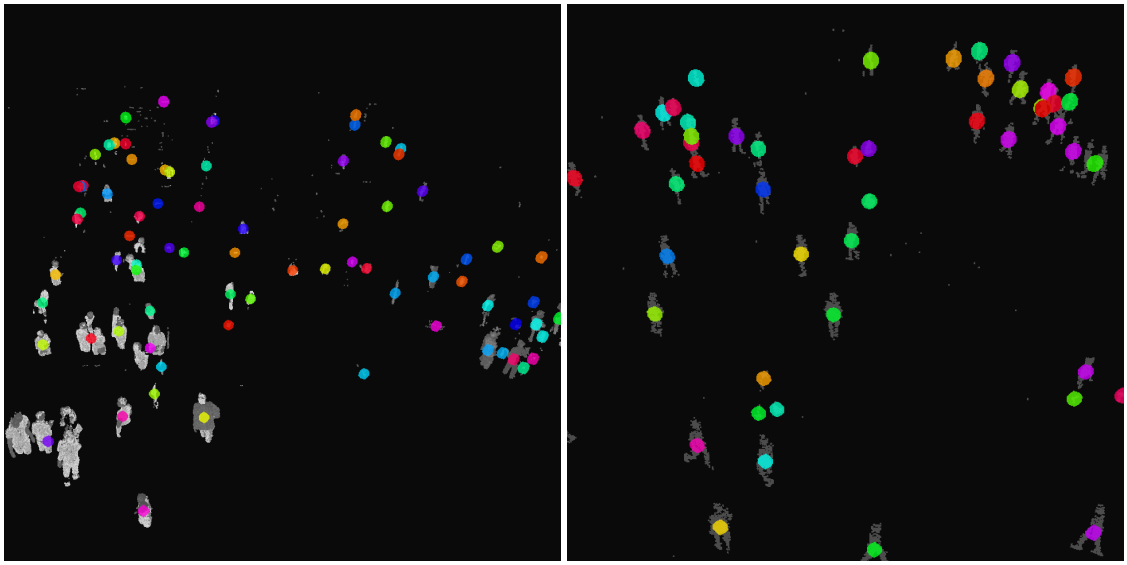


Figure 4.7: A visual representation of human clustering on real data (left) and simulated data (right). The coloured dots illustrate the centroid of each cluster.

4.3 Head Clustering Approach

In this section, the results from the head clustering approach, described in section 3.3, are presented.

4.3.1 Finding Clustering Parameters

Figure 4.8 and Figure 4.9 show the results from clustering using varying parameters, `eps` and `min_points`. In Figure 4.8, the predictions are compared to the actual number of humans in the scene, while in Figure 4.9, they are compared to the number of heads in the scene. Since there are usually people whose heads are not visible in the image, these show somewhat different results. When comparing to the actual number of humans, Table 4.2 tells us that `eps` = 20.0 and `min_points` = 4.0 give the lowest mean absolute error at 44.19. The same combination of `eps` and `min_points` gives the lowest mean absolute error when comparing to the number of heads visible as well, with a value of 7.82, as seen in Table 4.3. Since both tests gave the same parameter values, those were chosen for the evaluation. However, if they would have differed, the evaluation should be done based on the actual human count to get the most accurate results.

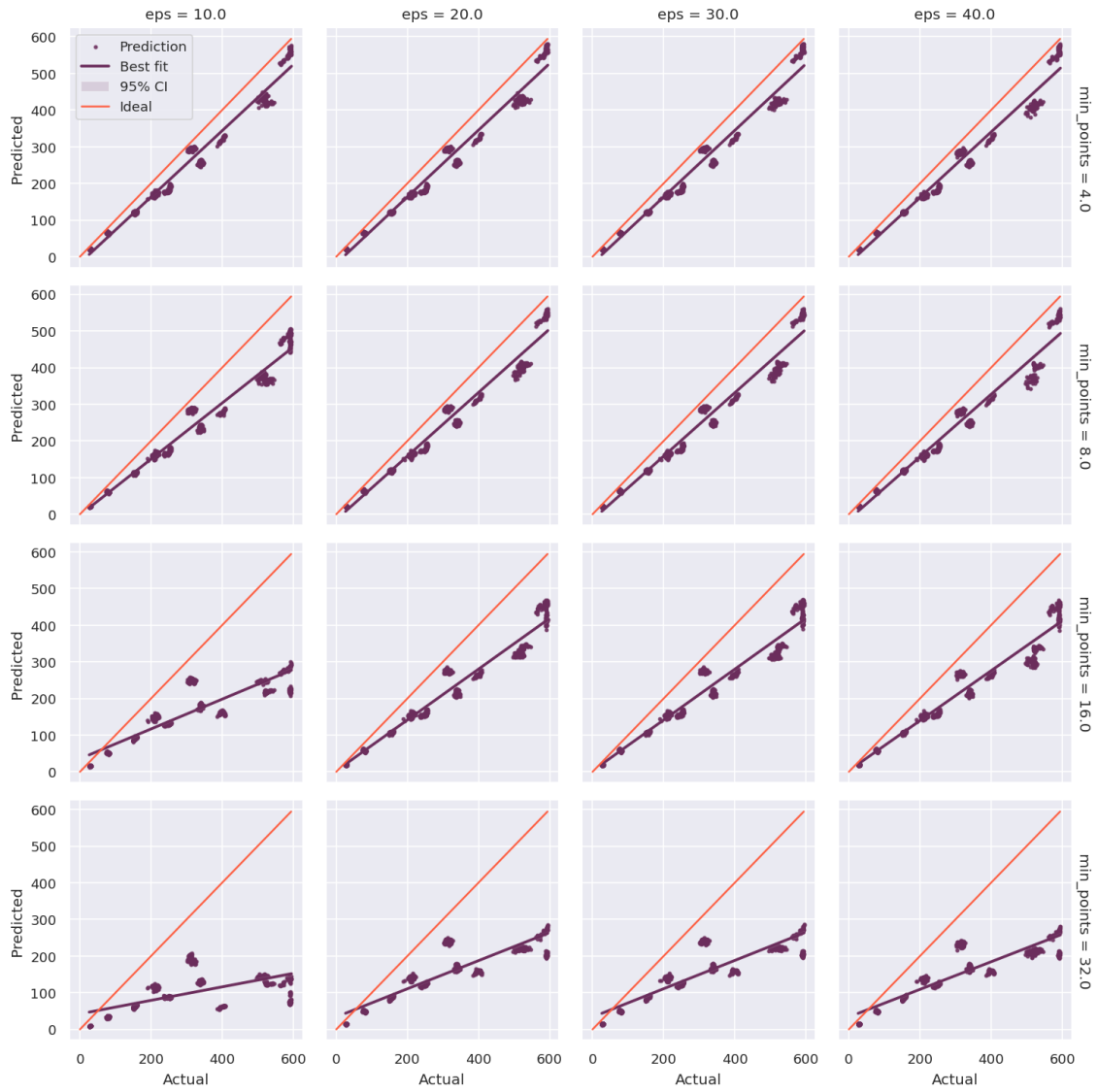


Figure 4.8: Regression estimates for all combinations of `eps` and `min_points`. The actual number represents the number of humans in the scene.

Table 4.2: Mean absolute error for all combinations of `eps` and `min_points` for Figure 4.9.

	eps = 10.0	**eps = 20.0**	**eps = 30.0**	**eps = 40.0**
min_points = 4.0	44.93	44.19	44.53	47.48
min_points = 8.0	68.95	52.37	52.63	55.55
min_points = 16.0	137.07	83.93	83.90	87.77
min_points = 32.0	188.19	140.18	139.81	142.36

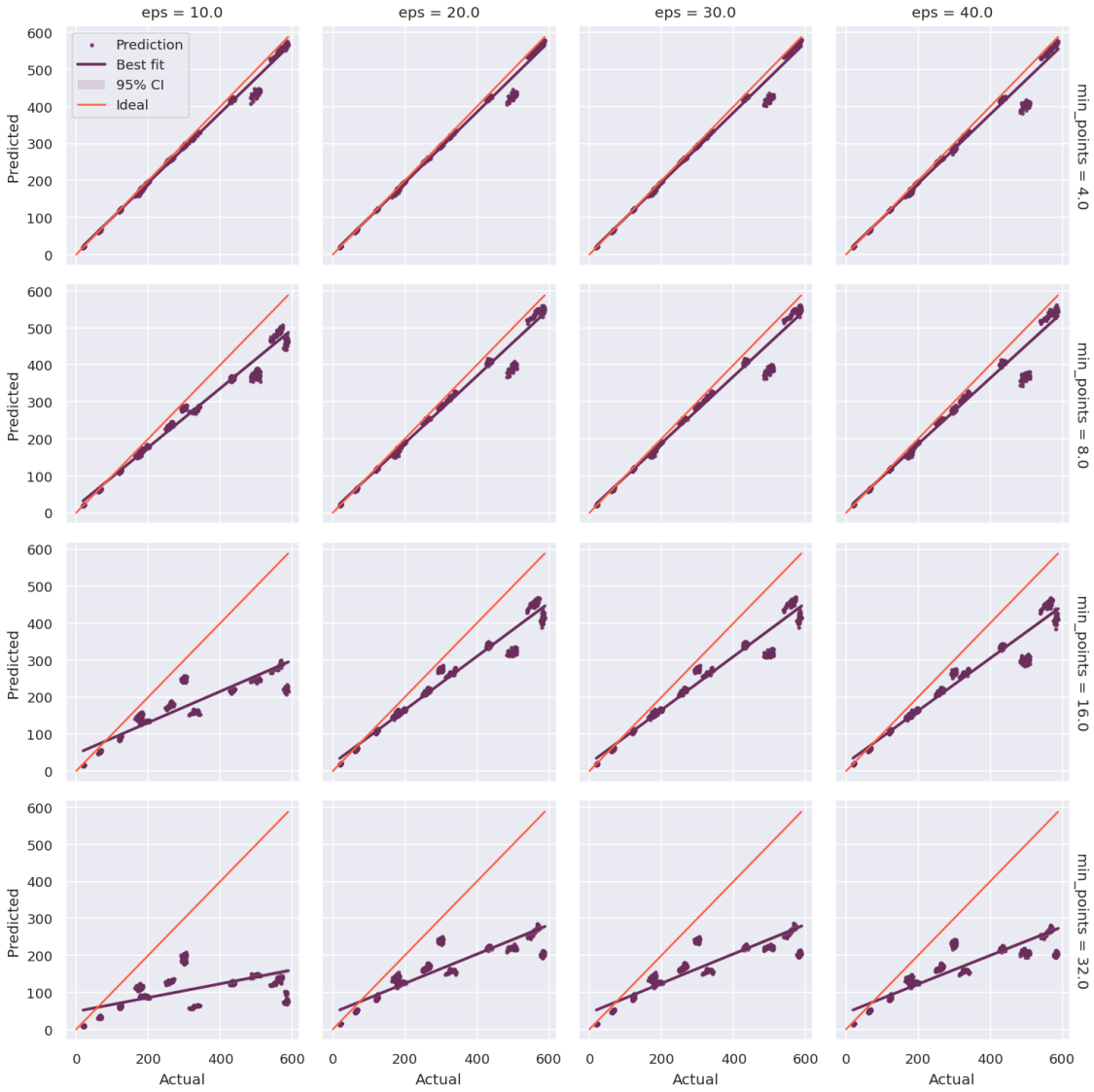


Figure 4.9: Regression estimates for all combinations of `eps` and `min_points`. The actual number represents the number of heads visible in the scene.

Table 4.3: Mean absolute error for all combinations of `eps` and `min_points` for Figure 4.9.

	eps = 10.0	**eps = 20.0**	**eps = 30.0**	**eps = 40.0**
min_points = 4.0	8.62	7.82	8.15	11.11
min_points = 8.0	32.58	16.00	16.25	19.18
min_points = 16.0	95.70	47.56	47.53	50.39
min_points = 32.0	151.81	103.81	103.43	105.98

4.3.2 Evaluation

The results from running the evaluation datasets through the head clustering pipeline are presented below.

Figure 4.10 shows the results from the quantity test. The predictions diverge from the ideal line as the number of people in the crowd increases, showing poor performance for crowds larger than 100 people with a mean absolute error of 67.87 with respect to all data. When looking closer at the predictions made on crowds with less than 100 people, it shows better performance with a mean absolute error of 9.70, as can be seen in Figure 4.11.

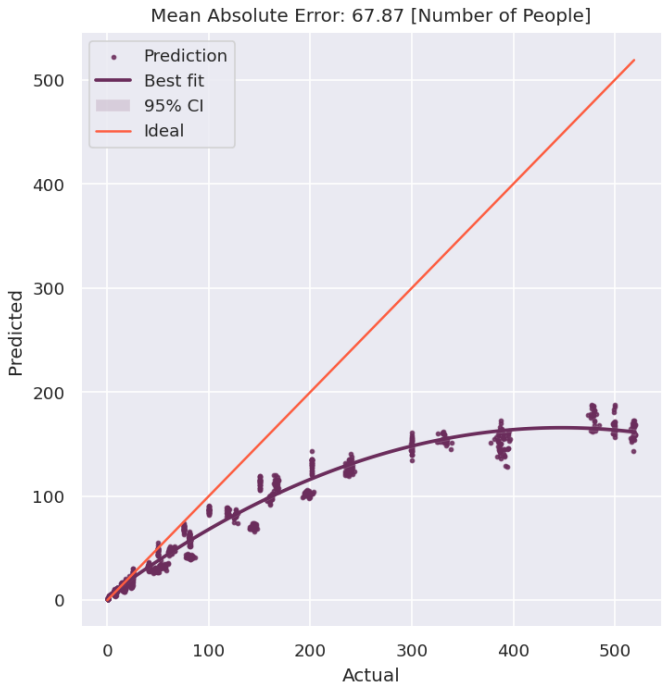


Figure 4.10: Quantity test results from the head clustering approach.

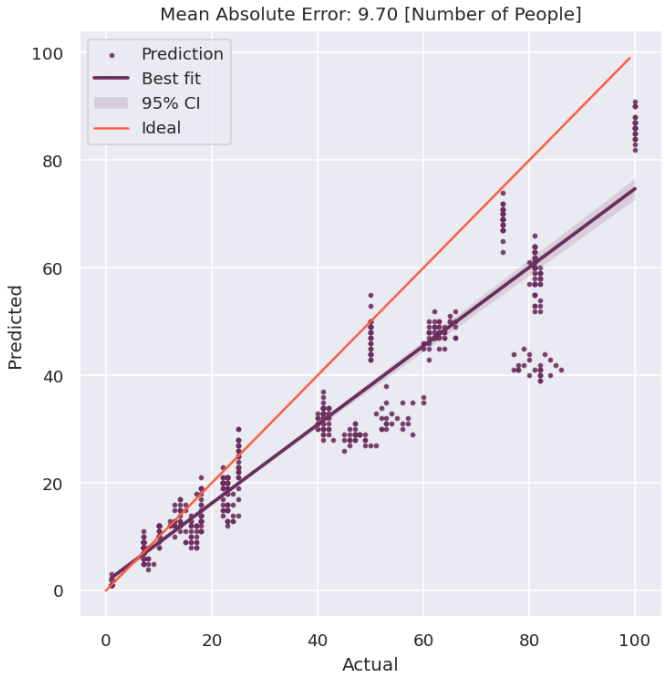


Figure 4.11: Quantity test results from the head clustering approach for crowds with less than 100 people.

When performing the density test on the head clustering approach, it is clear that it performs better with less dense crowds. In Figure 4.12 (a), the error gets closer to zero when the area is increased. However, it never reaches zero, which matches the results shown in the quantity test, where the model always estimates lower than the actual number of people. The model does not seem to gain confidence as the density decreases, as the distance between the upper and lower bounds of the whiskers stays relatively the same throughout the test.

For Figure 4.12 (b), on the other hand, the clustering results were almost perfect. However, even here it is possible to see an improvement when the density of the crowd increases.

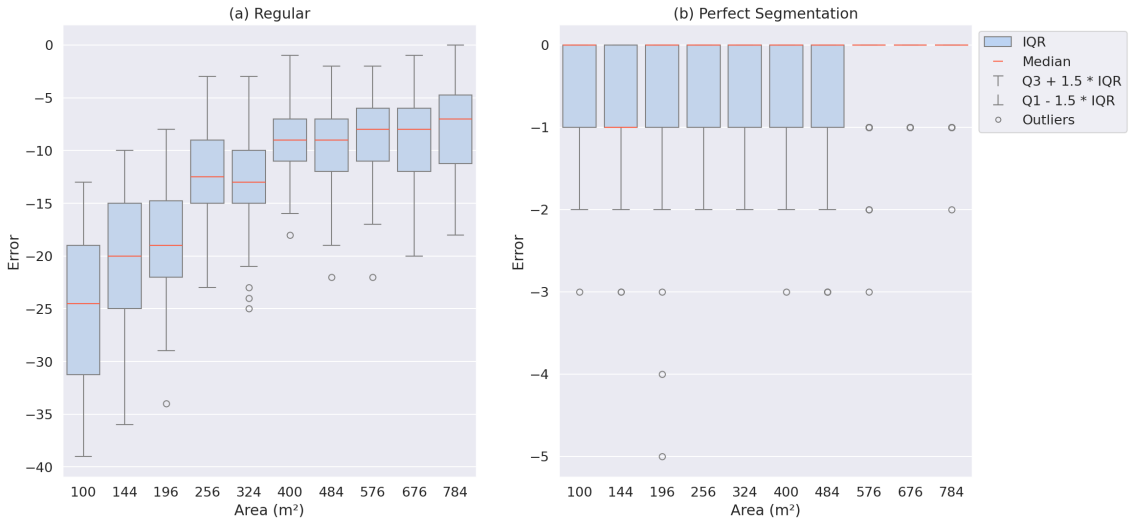


Figure 4.12: The density test results from the head clustering approach. (a) shows the actual results achieved on the output from the segmentation model, while (b) is head clustering performed on perfectly segmented heads. Achieved by clustering the annotated segmentation data.

Figure 4.13 shows the results from the distance tests. It can be seen that the head segmentation model performed best when the LiDAR was as close as possible. As it was moved further away, the performance quickly became worse, and the model became less confident. However, there are multiple outliers centred around -40 error when the distance is low; this will be discussed further in subsection 5.3.2.

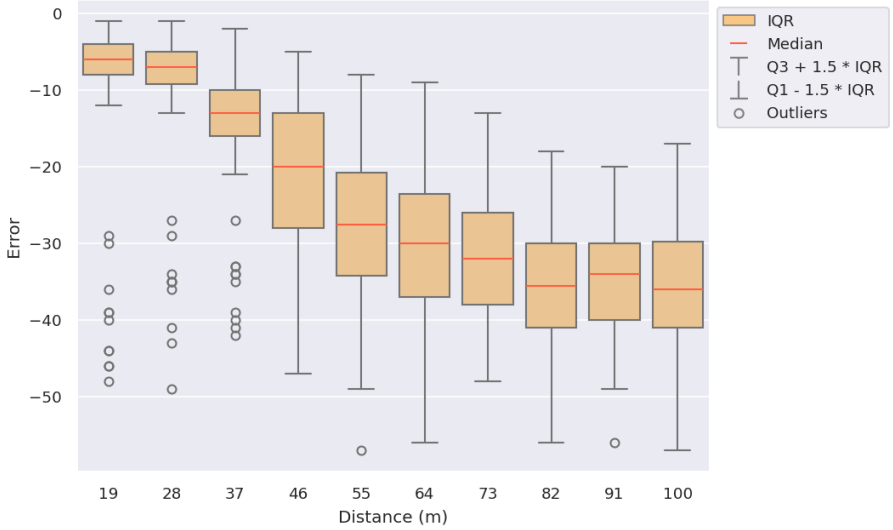


Figure 4.13: Distance test results from the head clustering approach.

The clustering results from the head-segmentation model on real data are shown in Figure 4.14 (left). Here it can be seen that the centroids of the clusters are located close to the heads of people. The model failed to segment the heads of some people, however, in some

cases, it could separate people standing close together. Furthermore, it did not count the noise at the top of the image as people.

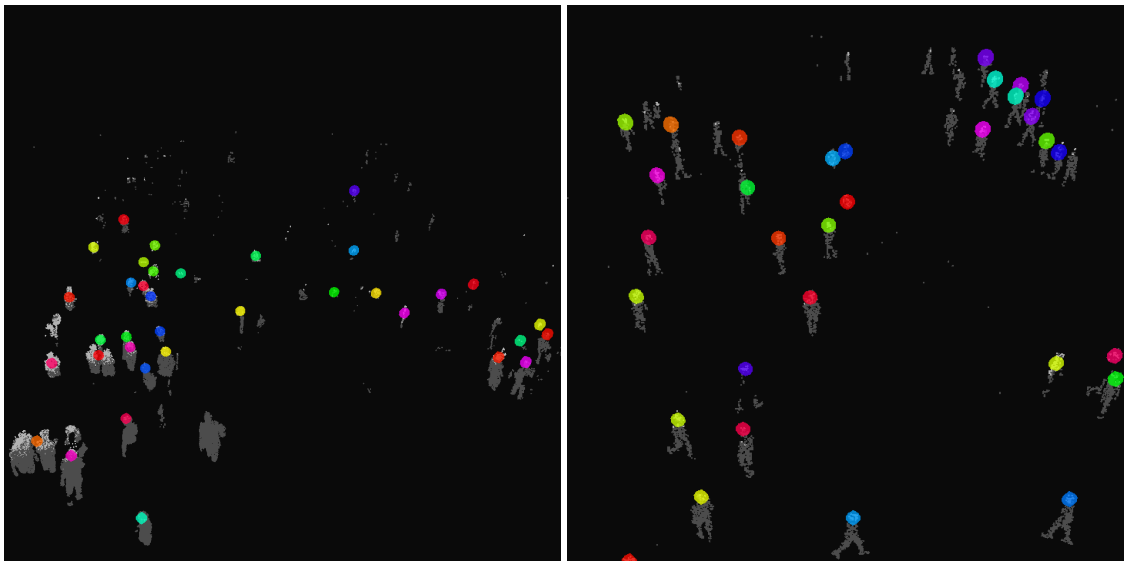


Figure 4.14: A visual representation of head clustering on real data (left) and simulated data (right). The coloured dots illustrate the centroid of each cluster.

4.4 Density Estimation Approaches

In this section, the results from the density estimation approaches described in section 3.4 are presented.

4.4.1 Evaluation

The quantity evaluation shows some correlation between the prediction of the number of people and how many people there actually were in the sample, as seen in Figure 4.15. The incline of the predictions from the class densities approach closely matches the ideal line, shifted up around 400. The predictions from the exact densities approach lie closer to the ideal line, while the incline is a worse match.

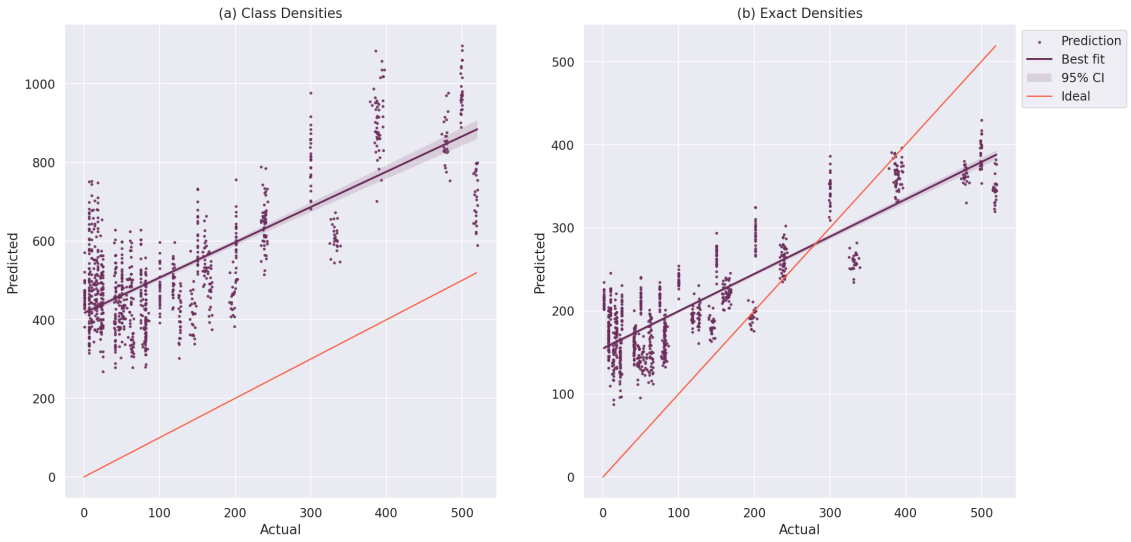


Figure 4.15: Quantity test results from the density estimation approaches, (a) Using classes as densities, (b) Using exact densities.

The results from the density evaluation show an interesting finding. Figure 4.16 show that the performance is worse when the crowd is spread out over a larger area, meaning that these models perform better with increased density. In the case of exact densities, it works best at a density of around 0.39 people per square metre, while the class densities approach works better with increased density.

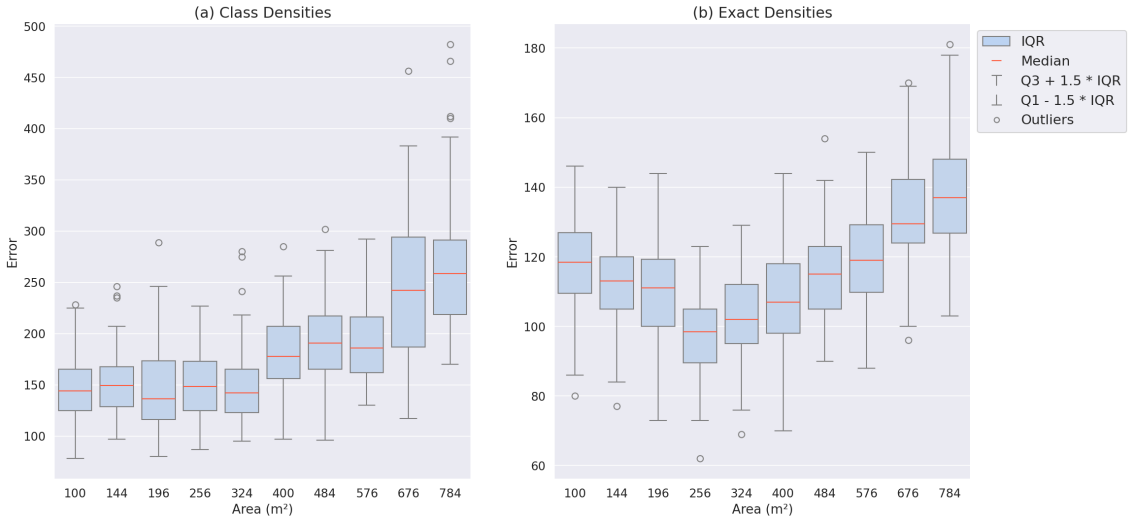


Figure 4.16: Density test results from the density estimation approach, (a) Using classes as densities, (b) Using exact densities.

In the distance evaluation test, both methods performed best when the distance between the LiDAR and the crowd was around 46 metres, as can be seen in Figure 4.17.

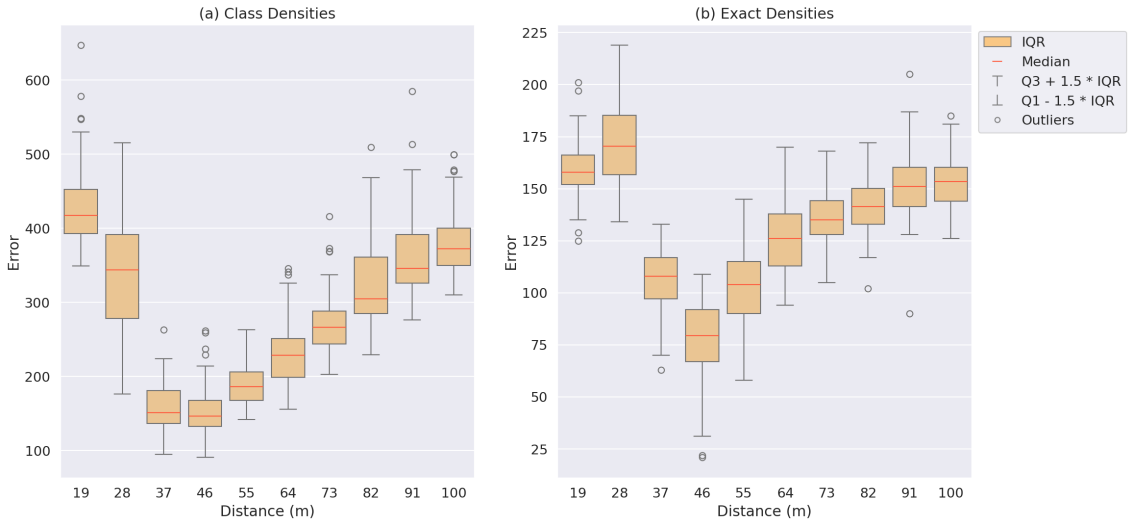
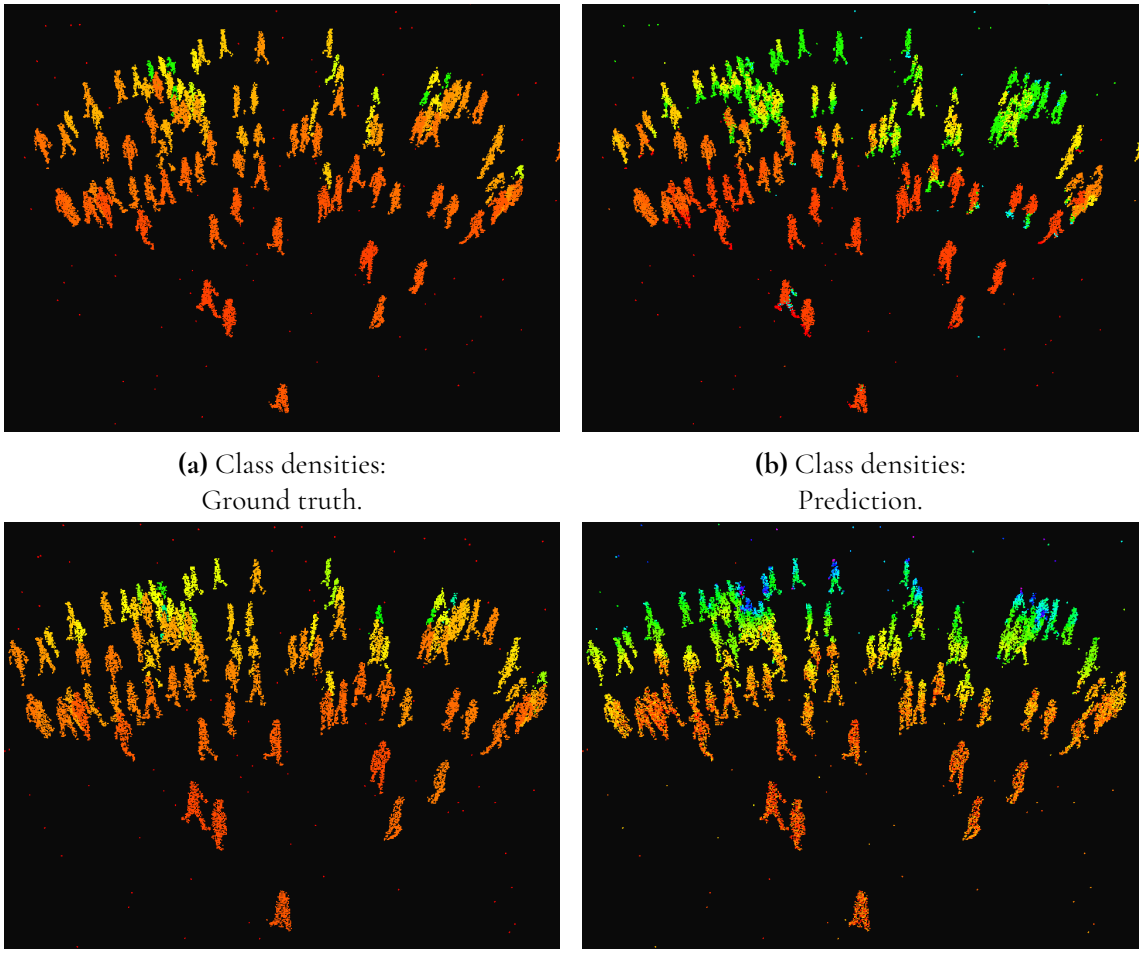


Figure 4.17: Distance test results from the density estimation approach, (a) Using classes as densities, (b) Using exact densities.

A comparison between ground truth and a prediction can be seen in Figure 4.18. The ground truths of both of the approaches show that each person is represented by a single colour, i.e. the same density for each point. For the class densities approach, there are only 20 different densities a point can have, therefore, it is common for different people to have the same colour. However, for the exact densities approach, each point has the exact density for a given person. The predicted result did not follow our desire that every person should consist of points with the same density, instead, multiple different densities can be found in the same person. Both approaches did also show signs that they predicted an excessively high density estimation for points that are far away from the camera, but points closer to the camera are more accurately predicted.



(a) Class densities: Ground truth.

(b) Class densities: Prediction.

(c) Exact densities: Ground truth.

(d) Exact densities: Prediction.

Figure 4.18: Visual representation of the class densities approach (a and b) and the exact densities approach (c and d). The colour spectrum goes from red to blue, where red represents a low density at a point and blue represents a high density at a point.

4.5 Deep Learning on Point Clouds

In this section, general results from the training process of the deep-learning models are presented. These will be used to discuss the applicability of deep learning on point clouds.

Figure 4.19 and Figure 4.20 shows the confusion matrices for the training and test datasets respectively. The confusion matrix given from the training dataset shows a promising diagonal line, while Figure 4.20 has more difficulties dealing with the middle classes from 7 to 13.

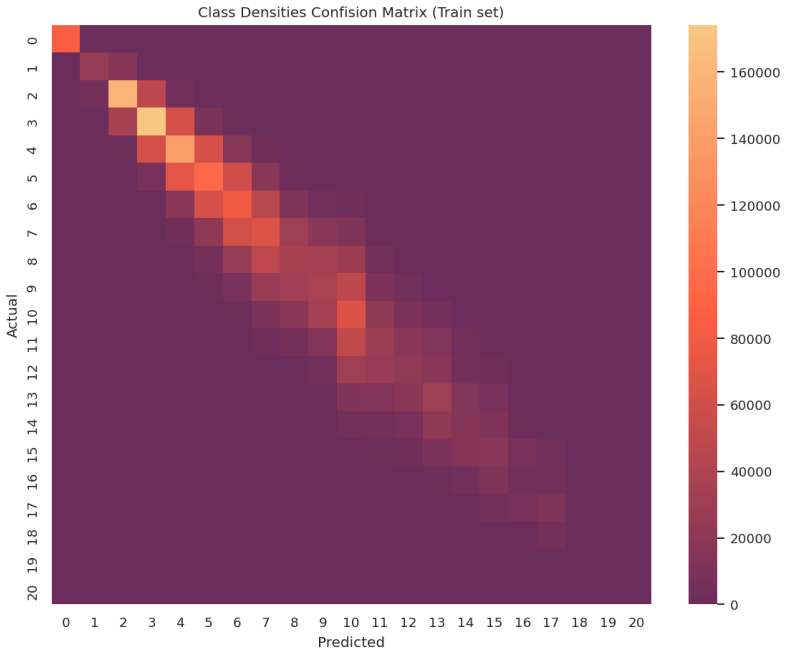


Figure 4.19: Confusion matrix from the train split of the training dataset for the class densities approach.

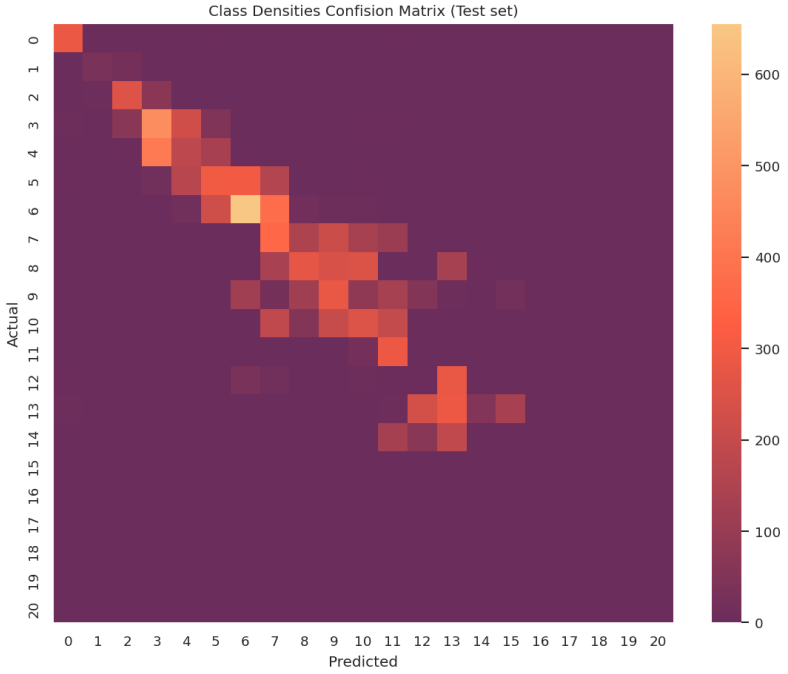


Figure 4.20: Confusion matrix from the test split of the training dataset for the class densities approach.

The f1-score from training the class densities approach is shown in Figure 4.21. This graph shows no indication of traditional overfitting.

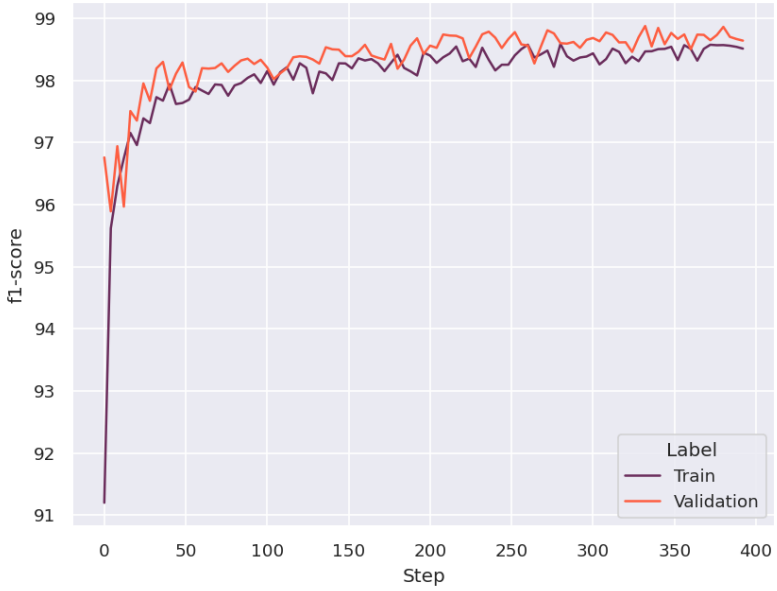


Figure 4.21: Training and validation graph for the class densities approach.

From Table 4.4, the inference time for each approach is presented. The human clustering approach, which was the only approach that did not use deep learning, was fastest at 0.18 seconds, while the head clustering approach was the slowest.

Table 4.4: Inference time for the different approaches, in seconds.

Human Clustering	Head Clustering	Class Densities	Exact Densities
0.18s	12.66s	2.48s	2.37s

Chapter 5
Discussion

The following chapter is organised as follows: first, the datasets are discussed, followed by an examination of the results for each approach. Finally, an outline of the future research directions will be presented.

5.1 Dataset

In this section, we will examine our dataset. First, we will highlight the advantages of simulated data, followed by its limitations. Then, we will move on to real data, discussing its advantages and limitations.

5.1.1 Advantages with Simulated Data

The main reason why simulated data was used instead of real-life captured data was to minimise the time spent on recording and annotating data. In UE5, we were able to implement different ways of annotating data, enabling us to try all our approaches. For the human clustering approach, the data had to be annotated, such that every point got a class of 1 if it was a point on a human and a 0 otherwise. For the head clustering approach, only the heads should have an annotation set to 1. The two density estimation approaches were a bit different, since we needed to keep track of a person ID as well in order to give the points the correct class or float annotation.

When training a deep-learning model, it is important to have a diverse and large dataset. With a larger dataset, it is possible to use a more complex model with a reduced risk of overfitting [29]. The simulated training dataset consisted of 10,500 samples, totaling 97GB of raw data.

Another advantage of simulating data was the ability to record crowds in more locations than would have been possible in real life. As mentioned in subsection 3.1.1, we recorded

data in parks, cities, and in our own map. These different locations made our dataset more diverse, which is good for machine learning [29].

We could also make the data look how we wanted it to, meaning that we could decide how many people there were in each scene and how close together they should be. Additionally, we were able to place the LiDAR anywhere we wanted, not having to consider any physical limitations.

5.1.2 Limitations with Simulated Data

There are also some limitations to using simulated data. Limitations mostly concern the low entropy in simulated data and LiDAR physics. Creating realistic scenes with great diversity is a laborious task and requires variety in the 3D models. People should wear different-looking clothes with unique patterns and colours, and humans should have varying heights and sizes. Realistic scenes should also contain clutter, such as loose objects on the ground, vegetation, and vehicles. Crafting scenes that met this standard would require more time than was available within the scope of this master's thesis. Hence, the following restrictions were implemented:

- Few variations in humans
- No loose objects on the ground
- No weather simulation; only daytime with a clear sky

Despite these limitations, it was possible to simulate life-like crowds. The first limitation was that there were few variations in the appearance of humans. The non-playable characters we used to create a crowd, as described in subsection 3.1.1, called MetaHumans, were a collection of six differently-looking humans, including both males and females. The differences between them were height differences, clothing, and skin colour. Due to the lack of colour in LiDAR data, clothing and skin colour have little to no impact on the data. The height difference, however, is noticeable in LiDAR data, so we would have wanted more variation in this case, such as children or really tall people. Other variations that appear in a real-world scenario that we would have liked to include in our data include someone carrying a backpack, someone with a stroller, and someone in a wheelchair.

The second limitation is concerning loose objects on the ground. Lose objects that do not move will be background subtracted and will therefore not be visible in the data. Moving objects, on the other hand, will still be left in the data after background subtraction. However, both static and moving objects will induce occlusions.

Section 1.3 mentions that LiDARs tolerate bad weather and lighting, therefore, it does not matter whether the simulation was able to simulate weather or nights as well. Therefore, we did not create samples from both night and daytime.

To create LiDAR data from the simulated crowd, a LiDAR sensor was used. The LiDAR sensor we used in UE5 had some differences from a real LiDAR:

- Reflection intensity is not accurate
- No cone shaped rays
- LiDAR noise

Firstly, the LiDAR sensor was able to detect reflection intensity, however, for this to work well, all materials in the world had to be realistically designed. In our opinion, the available materials were not realistic enough, so we decided not to use reflection intensity as part of the data. We also had to disable laser bounces. This decision is related to the lack of realism in the reflections, because if the reflections are not accurate, the bounces will not be accurate either. Secondly, the LiDAR sensor did not simulate the cone effect of a real LiDAR ray, since line traces in UE5 are infinitely thin. A real LiDAR ray spreads with distance in a cone shape, and therefore it is possible for the light to go through foliage, which gives it the ability to "see through" bushes and grass. The simulated ray would only return the first hit, whatever it may be. Lastly, the LiDAR noise was not fully realistic. We chose not to prioritise creating realistic noise because it would take too long to find the parameters, and we felt like the noise would not impact the model positively or negatively. With this said, we did still have some noise in order to not make the data too perfect.

5.1.3 Advantages with Real Data

As described in subsection 5.1.2, we were not able to utilise reflection intensities in our training because of the lack of realistic materials in our simulation. If we had trained on real data instead, the reflection intensities would be accurate and therefore usable as features in our training data. As seen in Figure 4.1, it seems like clothing has a high reflection intensity, while skin has a low reflection intensity. This observation makes us believe that it would be easier to segment heads from bodies.

Real data would also result in a more diverse dataset with more variety in how people look and behave. This variety would improve the scope of the models we are training.

5.1.4 Limitations with Real Data

Recording LiDAR data manually would have required us to find locations with crowds and acquire permission to record said crowds. During this master's thesis, we were only able to record at one location and only for a few hours.

Another limitation with real data would be the difficult annotation process. Besides the fact that annotating is a time-consuming and tedious task, it is also difficult to make correct annotations. For example, by looking at Figure 4.1 (b), it is not possible to determine the number of people in the scene, especially the dense crowd that is further away. Maybe it would have been possible to take camera pictures from the same angle and pair them with the LiDAR samples. The combination of a camera and LiDAR would simplify the annotation process because it is easier to see details in an image compared to point clouds.

These limitations restricted our use of real data. Since we did not do any annotation of our real data, it was not possible for us to use it in the training phase for our deep-learning approaches. We were also limited in how we could evaluate approaches with real data, meaning that we were only able to do some visual evaluation, as in Figure 4.7 (left) and Figure 4.14 (left).

5.2 Deep Learning on Point Clouds

Performing deep learning on point clouds has proven to be a difficult task. There are multiple challenges working with LiDAR data compared to regular images, such as invariant data size and the unordered structure of point clouds. During the process of this master's thesis, we wanted to try multiple different models besides PointNet++; however, we did not get any other models to train successfully.

When training the density estimation models, the resulting confusion matrices looked optimistic with a strong diagonal line, as seen in Figure 4.20. However, even though the line is diagonal, there is considerable bleeding into nearby classes. Errors in the central classes 7–13 have less impact on the resulting density map due to the nature of the class assignment function, described in Equation 3.3.

In Figure 4.21, the graphs do not indicate a traditional overtraining where the validation curve diverges from the training curve. Therefore, we believe the issue is that the validation samples are too similar to the train samples and do not give a good indication of how generalised the model is.

One way to improve the generalisation of the deep-learning models is to introduce data augmentations. Data augmentation helps increase the size of the dataset without adding new samples, in the sense that the same samples can contribute more when they are slightly altered each time. We have experimented with applying random mirroring along the x and y axes without any apparent effect on the results. An augmentation that we did not have time to explore was random jitter. This is when all the points in the point cloud are nudged a small distance in a random direction. This could help with generalising for the noise that comes naturally when filming with a LiDAR sensor. The fixed-point transform could also be improved so that it minimises loss of data. When downsampling, we could choose to remove points based on how far away they are from the sensor, which would result in a point cloud that is more evenly distributed. Since objects close to the sensor are more densely represented compared to objects further away, we can afford to lose more of those points.

To improve the head segmentation model, we believe that larger networks would be useful. The network used in this thesis consisted of around 1,700,000 trainable parameters. The task of segmenting only the points belonging to heads and not any other points requires the model to have high accuracy, which can be achieved by larger models. We have not experimented with any larger models since the hardware we used was not sufficient. When training the model, it consumed around 11–12GB of video memory out of the 12GB available on the NVIDIA GeForce RTX 3060 we used. We were therefore limited in both the network size and batch size we could use when training.

There is also a large difference in inference time between the approaches. Table 4.4 shows that the approaches based on deep learning took multiple seconds longer to count people compared to the human clustering approach, which did not use any neural networks. This

was an important finding, as it tells us that only the human clustering approach can be used in real time. Since we capture the LiDAR frames at a rate of 1Hz, the counting has to be done within one second. It should also be noted that the head clustering approach was slower than the other approaches. This was mainly because of the multiple inferences per sample, described in section 3.3, to even out the predictions by the model. When testing the model, we set it to predict each sample five times, which explains the roughly five times longer inference time.

5.3 Approaches

This section will go through the four approaches and discuss the results. We end by comparing them to come to a conclusion about which methods show the most promise for future implementations.

5.3.1 Human Clustering

Even though the best fit was close to the ideal line in Figure 4.3, it does not mean that each cluster represents a human. There could be instances where two or more people are in such close proximity that they get grouped together within the same cluster. On the other hand, there could also be instances where noise is misclassified as clusters or where a single human is grouped into multiple different clusters. This type of under- and over-counting of clusters can be seen in Figure 4.7. The group of people in the bottom left of the top image are clustered into a single cluster, and multiple clusters of non-humans can be seen at the top of the same image. In Figure 4.7 (right), there are also examples of when a single person is divided into multiple clusters. In this case, the clustering led to an accurate number of people, even though the visual result was inaccurate.

Another reason why the best fit in Figure 4.3 was close to the ideal line could be because we chose parameters with respect to the ideal line of our train data. This might have led to overfitting. As visualised in Figure 4.2, the selection of `eps` = 30.0 and `min_points` = 8.0 resulted in the lowest mean absolute error for our training data. However, for other datasets, another combination of the parameters could have resulted in a better regression line. Despite this, when evaluating using the optimal parameters from the train dataset on the quantity evaluation dataset, the mean absolute error was still as good, indicating that the chosen parameters were valid.

The mean absolute error was 10.99 on the quantity evaluation dataset, while it was 24.41 for the training dataset. Furthermore, when only observing samples with 100 people or less, the mean absolute error was even lower, at 1.95. The reason why human clustering performed better on the evaluation dataset compared to the training dataset could be because it is a totally different dataset, and therefore it cannot be expected to produce the same result. The smaller mean absolute error in Figure 4.5 indicates that the clustering algorithm was more accurate when dealing with fewer people. This is probably because there is more space around them when there are fewer people, leading to a more spread-out crowd and hence an easier clustering task.

Moving on to Figure 4.5, showing the result from the densities test. The observation that the median error was consistently positive implies that there were consistently more clusters

detected than there should have been. It is especially interesting to note that the largest error was found when the area was the smallest. The fact that the error was positive correlates with Figure 4.4, where most of the samples of the actual 100 people were also overestimated. However, it seems more likely that the number of clusters will decrease as the density of people increases because there will be more possibilities for groups of people to be clustered together. The reason could instead be partial occlusions, where a person might be divided into two parts, e.g. when standing behind a light post or behind another human.

Figure 4.5 does also show that the interval between the whiskers is getting smaller with a larger area. This means that the spread of the error is smaller when the area is larger. This indicates that the clustering algorithm is more consistent when the people are more spread out.

For the distance test shown in Figure 4.6, it is clear that the error becomes larger when the distance increases. More specifically, we found that there were fewer clusters detected when the distance to the crowd was greater than 73 metres. The clustering algorithm had `min_points` of 8.0, meaning that a cluster needs to contain at least 8 points in order to be classified as a cluster. However, when recording a crowd from a long distance, it is possible for a human to be represented by fewer than 8 points. Additionally, the partial occlusions that occur in crowds are also a cause that decreases the number of points per human.

5.3.2 Head Clustering

An important finding in the results for the head clustering approach was the substantial deviation in the quantity test, seen in Figure 4.10. The accuracy of the model quickly decreases as the number of people increases. This could be caused by multiple different factors, such as the density increasing and the number of points remaining after the background subtraction increasing. Firstly, as we used a fixed size for the recording area, 50 by 50 metres, the density increased with the number of people. We know from the density test, Figure 4.12 (a), that the performance gets worse with increased density. However, the highest density achieved in the quantity test was around 5 square metres per person, which should mean an error of around $\frac{-8}{100} * 500 = 40$ according to the density test. Secondly, the large recording area also implied a large distance to the LiDAR. Calculating the error this brings is more difficult than the density error since the crowd is spread out over a larger area. When the number of people was 500, the distance to the closest person was only a few metres, and the furthest was around 70 metres away. Taking an average of 35 metres gives an error of around $\frac{-13}{100} * 500 = 65$ according to the distance test in Figure 4.13. These two observations make up for approximately 35% of the total error when there are 500 people in the crowd. Finally, the remaining error could be explained by the fixed-point transform. As the number of people increases, the number of points remaining after the background subtraction also increases. This means that the fixed-point transform has to remove more points to keep a constant number of points in each sample. Since the background is removed before the transforms are applied, all points removed by the fixed-point transform belong to humans, making it more difficult for the model to segment the heads.

One of the theorised strengths of the head clustering technique was that it would perform better in dense crowds compared to human clustering. This was believed because heads are usually further apart than bodies, which would be the case if the head segmentation model performed perfectly. However, as the density increased, the model predicted fewer and fewer

people, and the resulting segmentations contained more than just heads. In Figure 4.14, the lighter grey dots show which points have been labelled as heads. This decrease in segmentation performance could be explained by occlusion. When the crowd gets too dense, the only part visible to the LiDAR is the top of the heads of people. This means that the shape of humans to the LiDAR is completely different in dense crowds compared to less dense crowds. Even though the dataset did contain both types of crowds, the model may not have learned to recognise all the shapes a human may have. We believe that this could be drastically improved by incorporating more dense crowds into the dataset as well as increasing the entropy. An increase in the size of the machine-learning model could also allow it to learn more complex patterns in the data and perhaps help it recognise different human shapes.

The distance test also indicated a decrease in performance as the distance increased, as seen in Figure 4.13. The model is shown to perform best when the crowd is as close to the LiDAR as possible. As the distance increases, the people in the crowd make up fewer and fewer points, making it difficult for the segmentation model to find the heads. Therefore, it may be beneficial to record with a lower frame rate when the crowd is far away, allowing the LiDAR to capture more data in every point cloud. Another observation in the distance test was the outliers present at the distances of 19–37 metres. These seemed to fall into the range of the whiskers when the distance increased. The cause of these is likely that those samples came from a LiDAR with a suboptimal view of the scene. Since we used multiple LiDARs at once when recording the evaluation dataset, some of them may have had worse conditions for crowd counting or had a perspective not present in our training dataset.

When comparing how well the head segmentation works between real data and simulated data, it is clear in Figure 4.14 that heads are more exactly segmented for simulated data than real data. For real data, our segmentation model was able to somewhat successfully segment the heads. However, the model did also classify other body parts as heads, such as shoulders, therefore eliminating the advantage the head segmentation was supposed to have.

5.3.3 Density Estimation

The density estimation approaches did not perform as well as we had hoped. Even though performs well for camera-based methods, the way we implemented it for LiDAR was not optimal. All tests show that they predict unreliably, however, they still show some form of correlation, meaning that they have learned something from the training. One of the more interesting observations was how both of these models performed better with increased density, as shown in Figure 4.16. Since these methods do not rely on clustering, they do not necessarily have to be affected by the distances between the individual people. While it is difficult to know exactly what a machine-learning model learns during training, it is notable that it makes use of people standing close together to make its predictions.

In Figure 4.18, it can be seen what the models output. Looking at these images, it is clear that this is not a good way to represent a density map in 3D data. Instead of getting an overview of the crowd distribution, the densities outputted from our models simply represent how densely the person has been composed of points or how far away they are from the LiDAR. This more closely resembles a regression-based method than a density estimation method, since the intermediate product is of little value as a density map.

There are multiple reasons why we believe these approaches did not perform well, such as suboptimal machine-learning models, training data, and flaws in our implementation.

Firstly, the deep-learning models we used were designed for segmentation problems. Although we believed that density estimation was similar enough to segmentation for it to be substitutable, there are some differences that we did not think through. The output density maps from camera-based models are usually of lower resolution compared to the inputs, meaning that a single pixel in the density map covers a larger section of the image. When viewing the problem as a segmentation task, the output is not scaled down, and the densities only cover a single point in a point cloud. Another difference is that point clouds are sparse, while regular images are dense. There can be arbitrary distances between points in a point cloud, while image pixels are on a fixed grid. This means that densities in regular density maps can be spread out over multiple pixels, which is not always possible in point clouds. A possible way to solve these issues is by transforming the point clouds into voxel grids and performing regular CNN operations to find the features of the point clouds. By doing this, the voxel representation of the point clouds would be dense, and we could downscale the output, solving both issues with density maps in point clouds. The downside to this technique is that it would be highly data-intensive since the voxels would have to be small to not lose information.

Secondly, the training data was not optimal for the density estimation approaches. For density estimation to make sense, some areas of the input should be able to have zero density. In our data, the only objects in the scene are humans, meaning that all points in the input had density. Therefore, the model never saw any samples containing zero density and may not have learned how they should be represented. To improve on this, other moving objects that pass through the background filtration, such as cars and hand baggage, should be introduced in the training data. It may also be interesting to skip the background filtration step, however, that would increase the point cloud size drastically.

Lastly, our implementation of density estimation on point clouds was flawed. To represent densities using classes was not optimal, as the difference between the classes was too small for the model to accurately distinguish between them. For instance, using the class assignment function shown in Figure 3.5 (a), the difference between classes 10 and 11 is only 20 points per person in the point cloud, with no difference in the overall shape of the person.

5.3.4 Comparing the Approaches

The results from the evaluation datasets clearly indicate that the human clustering approach outperforms all other proposed methods. Notably, human clustering is the only method that does not incorporate deep learning at any stage of its pipeline. In contrast, both the class densities approach and the exact densities approach, which heavily rely on deep learning, yielded the least accurate results. The head clustering approach, which combines the use of deep learning and clustering, performed better than the density approaches but still fell short of the human clustering method. This suggests that while incorporating deep learning increases the complexity of the problem, it also offers the potential for a more dynamic system. Despite the current shortcomings of our deep-learning approaches, particularly head clustering, they show promise for improvement.

Even though human clustering generates the best results, it is difficult to recommend this approach as a trustworthy solution. As discussed in subsection 5.3.1, it seems that an accurate approximation of the crowd count was the outcome of mainly two clustering errors, namely, grouping together people in close proximity of each other and misidentifying background

noise as people. Therefore, in the case of crowd distribution analysis, we believe that the head clustering approach has the highest potential among the four proposed approaches.

As discussed in section 5.2, the human clustering approach was the only one that could predict the number of people in a crowd in real time. However, since the approaches only generate an estimate at best, it might not be needed for the implementation to be able to function in real time. Instead, an update of the crowd count every 12 seconds might be sufficient, at least for large crowds.

One promising finding about the density approaches compared to the other two approaches was that they did not lose accuracy when the crowd density increased. Both human clustering and head clustering depended heavily on clustering algorithms and were therefore impacted by the density of the crowd. The concept of head clustering was to increase the minimum distance between two people by only using their heads when doing the clustering. Our implementation of the head segmentation part of the head clustering approach did not perform well enough to be better than human clustering in dense situations. However, as seen in Figure 4.12 (b), head clustering with perfect head segmentation did perform better than human clustering, indicating that our concept was valid.

5.4 Future Work

Several intriguing ideas emerged during our research that we did not have the opportunity to explore fully. In this section, we will discuss these ideas with the hope of inspiring future research in this direction.

5.4.1 Alternative Approaches

Projecting point clouds onto 2D planes is a completely different approach compared to the ones explored in this thesis. This projection would make it possible to utilise other machine-learning techniques, such as regular CNNs and regression, to count people. Essentially, these projections could be seen as "taking pictures" of the point cloud that can later be fed into networks closely related to the ones described in section 2.1. An advantage of doing this on point clouds would be that it is possible to capture a single point cloud from multiple angles, all of which can be used to count, with the hope of extracting as much information from the point cloud as possible. Compared to the approaches explored in this thesis, this approach could utilise more popular and well-researched machine-learning techniques, possibly leading to better-performing models. However, the lack of annotated data to train such models means that simulated data still has to be used or that time-consuming manual annotation has to be done.

Another alternative approach is to estimate the danger level of a crowd rather than predicting the exact number of people. This method is less complex than people-counting, so a smaller model might suffice, and data annotation would be simpler. Although this approach provides less detailed analytical data, it remains useful for monitoring large crowds to assess overall safety.

5.4.2 Train with Real Data and Improve Simulation

As discussed in subsection 5.3.2, our approaches perform poorly on real data, likely because the models were only trained on simulated data. We believe that there is much to gain in performance by mixing in real data into the training dataset. We have not had the time to explore the impact of training on real data since the annotation process was deemed to be too time-consuming to be a part of this master's thesis.

5.4.3 Algorithmical Approaches

In this thesis, we have mainly focused on machine-learning-based approaches. This is not necessarily the optimal strategy. Instead, it might be interesting to study how other, purely algorithmical solutions may perform.

One such algorithmical approach may be to estimate the volume a crowd occupies and model a relationship between volume and people count. Parameters such as the density of the crowd will have an impact, however, it may produce a good estimate for homogeneous crowds with even density distributions.

An advantage of using algorithms compared to machine learning is that there is no need to annotate training datasets. However, some annotated data will still be needed to evaluate the performance. Since machine-learning approaches may be computationally slow and resource intensive, as for the approach discussed in section 2.3, an algorithmical approach may save computation time and energy consumption.

Chapter 6
Conclusion

Counting people in crowds using point-cloud-based techniques has proven to be both challenging and interesting. The primary aim of this thesis was to devise a solution for accurately counting individuals in crowded settings, thereby enhancing safety and preventing potential hazards. Four approaches were developed and implemented, showcasing promising results in different ways. These approaches were evaluated based on three key principles: the number of people, crowd density, and distance from the LiDAR sensor. Both the data used for evaluation and the training data were made synthetically in UE5. Additionally, some visual assessment has been conducted on real LiDAR data.

To answer **RQ1**, the human clustering approach demonstrated the most robust overall performance, accurately counting crowds of up to 500 individuals with an average error of only 11 people. While the deep-learning models showed potential, further refinement of both data quality and model architecture is necessary to achieve improved results. Although the head clustering approach shows promise, particularly in dense crowds, enhancements to the segmentation model are required to fully capitalise on its potential. However, our implementation of density estimation approaches gave disappointing results due to flaws in the implementation. An alternative method for implementing density estimation has been suggested as a potential extension of our work.

Regarding **RQ2**, we recommend continued exploration of point-cloud-based crowd counting using deep learning, as it shows promising indications of being a viable solution. In particular, the head segmentation approach demonstrates potential for improved performance in dense crowds compared to human clustering, provided that the segmentation model is improved.

References

[1] Nazarizal Mohammad. *Sunset over Bungkarno Stadium, Jakarta*. URL: https://unsplash.com/photos/aerial-view-of-city-during-daytime-abqlAKDYxxA (visited on 10/06/2024).

[2] John Crace. "Astroworld: deaths of 10 people at Houston concert ruled accidental". In: *The Guardian* (). URL: https://www.theguardian.com/music/2021/dec/16/astroworld-festival-deaths-ruled-accidental (visited on 01/26/2023).

[3] Ninad Mehendale and Srushti Neoge. "Review on lidar technology". In: *SSRN Electronic Journal* (2020). DOI: 10.2139/ssrn.3604309.

[4] Annurag P S and Srinivaas A. "Advancements and applications of lidar technology in the Modern World: A Comprehensive Review". In: *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)* (July 2023). DOI: 10.1109/iceccme57830.2023.10252481.

[5] Akshita Patwal et al. "Crowd counting analysis using Deep Learning: A Critical Review". In: *Procedia Computer Science* 218 (2023), pp. 2448–2458. DOI: 10.1016/j.procs.2023.01.220.

[6] Zelong Liu et al. "Foreground Segmentation-Based Density Grading Networks for Crowd Counting." In: *Sensors (14248220)* 23.19 (2023), p. 8177. ISSN: 14248220. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=a9h&AN=172987315&site=eds-live&scope=site.

[7] Zhengxia Zou et al. "Object detection in 20 years: A survey". In: *Proceedings of the IEEE* 111.3 (Jan. 2023), pp. 257–276. DOI: 10.1109/jproc.2023.3238524.

[8] Zhuofan Zong, Guanglu Song, and Yu Liu. "Detrs with collaborative hybrid assignments training". In: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)* (Oct. 2023). DOI: 10.1109/iccv51070.2023.00621.

[9] Bo Wu and R. Nevatia. "Detection of multiple, partially occluded humans in a single image by bayesian combination of Edgelet part detectors". In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1* (2005). DOI: 10.1109/iccv.2005.74.

[10] Di Kang, Zheng Ma, and Antoni B. Chan. "Beyond counting: Comparisons of density maps for crowd analysis tasks—counting, detection, and tracking". In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.5 (May 2019), pp. 1408–1422. DOI: 10.1109/tcsvt.2018.2837153.

[11] Yunpeng Tai. "A Survey Of Regression Algorithms And Connections With Deep Learning". In: *CoRR* abs/2104.12647 (2021). arXiv: 2104.12647. URL: https://arxiv.org/abs/2104.12647.

[12] Antoni B Chan and Nuno Vasconcelos. "Bayesian poisson regression for crowd counting". In: *2009 IEEE 12th International Conference on Computer Vision* (Sept. 2009). DOI: 10.1109/iccv.2009.5459191.

[13] Sifatul Mostafi, Taghreed Alghamdi, and Khalid Elgazzar. "A bayesian linear regression approach to predict traffic congestion". In: *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)* (June 2021). DOI: 10.1109/wf-iot51360.2021.9595298.

[14] Bo Li et al. "Approaches on crowd counting and density estimation: a review." In: *Pattern Analysis and Applications* 24.3 (2021), pp. 853–874. ISSN: 1433-7541. URL: https://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edssjs&AN=edssjs.3285B501&site=eds-live&scope=site.

[15] Viresh Ranjan, Hieu M. Le, and Minh Hoai. "Iterative Crowd Counting". In: *CoRR* abs/1807.09959 (2018). arXiv: 1807.09959. URL: http://arxiv.org/abs/1807.09959.

[16] Nitin Kumar Saini and Ranjana Sharma. "Deep Learning Approaches for Crowd Density Estimation: A Review." In: *2023 12th International Conference on System Modeling & Advancement in Research Trends (SMART), System Modeling & Advancement in Research Trends (SMART), 2023 12th International Conference on* (2023), pp. 83–88. ISSN: 979-8-3503-6986-1. DOI: 10.1109/SMART59791.2023.10428557.

[17] Vishwanath A. Sindagi and Vishal M. Patel. "A survey of recent advances in CNN-based single image crowd counting and density estimation". In: *Pattern Recognition Letters* 107 (2018). Video Surveillance-oriented Biometrics, pp. 3–16. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2017.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0167865517302398.

[18] Ravil Mussabayev and Rustam Mussabayev. *Optimizing K-means for Big Data: A Comparative Study*. 2023. DOI: https://doi.org/10.48550/arXiv.2310.09819. arXiv: 2310.09819 [cs.LG].

[19] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[20] Sonya Coleman, Dermot Kerr, and Yunzhou Zhang. "Image Sensing and Processing with Convolutional Neural Networks". In: *Sensors* 22.10 (2022). ISSN: 1424-8220. DOI: 10.3390/s22103612. URL: https://www.mdpi.com/1424-8220/22/10/3612.

[21] Gabriela Csurka, Riccardo Volpi, and Boris Chidlovskii. *Semantic Image Segmentation: Two Decades of Research*. 2023. DOI: https://doi.org/10.48550/arXiv.2302.06378. arXiv: 2302.06378 [cs.CV].

[22] R. Qi Charles et al. "PointNet: Deep Learning on point sets for 3D classification and segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017). DOI: 10.1109/cvpr.2017.16.

[23] Charles R. Qi et al. *PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space*. 2017. arXiv: 1706.02413 [cs.CV].

[24] Epic Games. *We make the engine. You make it Unreal.* 2024. URL: https://www.unrealengine.com/en-US.

[25] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[26] Jacob Berntsson and William Winberg. *Pedestrian detection and tracking in 3D point cloud data on limited systems*. 2021.

[27] Robin Bernståle and Hjalmar Lind. *Segmentation, Classification and Tracking of objects in LiDAR Point Cloud Data Using Deep Learning*. 2021.

[28] Epic Games. *Create simulations that feel (almost) like real life*. 2024. URL: https://www.unrealengine.com/en-US/uses/simulation.

[29] Li Shen et al. *On Efficient Training of Large-Scale Deep Learning Models: A Literature Review*. 2023. DOI: https://doi.org/10.48550/arXiv.2304.03589. arXiv: 2304.03589 [cs.LG].

Appendices

EXAMENSARBETE Point-Cloud-Based Crowd Counting
STUDENTER Isak Jakobsson, Jonathan Runeke
HANDLEDARE Maj Stenmark (LTH)
EXAMINATOR Elin Anna Topp (LTH)

Personräkning i punktmoln, för säkrare folksamlingar

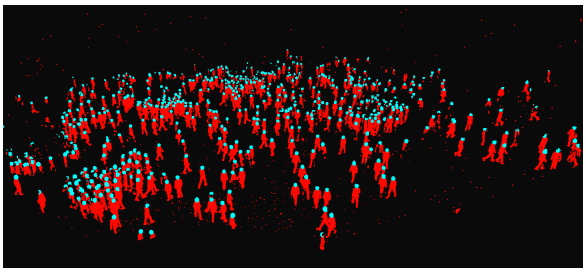
POPULÄRVETENSKAPLIG SAMMANFATTNING **Isak Jakobsson, Jonathan Runeke**

I takt med att allt fler människor samlas på offentliga platser ökar behovet av att snabbt och effektivt kunna räkna personer i folkmassor. Detta arbete undersöker nya punktmolnsbaserade metoder för att öka säkerheten och samtidigt bevara människors integritet.

Att räkna personer i stora folkhav, som på konserter eller under rusningstid på en stationsperrong, är viktigt ur flera perspektiv. Arrangörerna till konserten kan i god tid förhindra skaderisker och panikattacker, och på tågstationen kan man förutse olyckor och därmed stoppa flödet till perrongen och minska risken att någon knuffas ner på spåret. Genom att använda en LiDAR för att analysera utsatta områden ökar också integriteten för individerna i folkhavet, då personer i allmänhet är oidentifierbara i LiDAR-data.

LiDAR är en förkortning av *Light Detection and Ranging*, vilket översätts till ljusdetektion och avståndsmätning. Sensorn fungerar genom att skicka ut laserpulser för att mäta avstånd och skapa tredimensionella kartor av omgivningen, så kallade punktmoln.

I detta examensarbete har fyra olika metoder utvecklats och testats för att identifiera om det är möjligt att räkna personer med LiDAR. Tre av metoderna var baserade på djupa neurala nätverk och en byggde på en klassisk maskininlärningsalgoritm. För att träna nätverken framställdes syntetisk data från spelutvecklingsmjukvaran Unreal Engine 5. Metoderna utvärderades sedan utifrån deras prestanda med avseende på antal personer, folktäthet och avstånd till LiDAR-sensorn.



Resultatet visade att den klassiska maskininlärningslösningen är skalbar upp till 500 personer, men att den hade problem med att uppskatta en korrekt fördelning av folkmassor när tätheten blev för hög. Metoderna som använde sig av djupa nätverk gav intressanta resultat, men har många förbättringsmöjligheter. Bilden visar hur en av metoderna är tränad att identifiera huvuden.

Vi rekommenderar vidareutveckling av punktmolnsbaserad personräkning med LiDAR, med fokus på att förbättra modellerna med djupa neurala nätverk. Mer och bättre träningsdata skulle kunna leda till mer robusta och pålitliga system för att hantera folkmassor på ett enkelt och säkert sätt. Med sådana system kan vi vänta oss ökad säkerhet vid stora evenemang och offentliga platser, utan att kompromissa med individers integritet.