Pose classification of people using high resolution radar indoor

Anton Almqvist, Anton Kuusela

Master's thesis
2022:E29

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Lunds Tekniska Högskola

Master's Thesis

Pose classification of people using high resolution radar indoor

Anton Almqvist & Anton Kuusela

June 3, 2022

Supervisors:
Santhosh Nadig
Anders Skoog
Alexandros Sopasakis[LU]

LUNDS UNIVERSITET

LTH

LUNDS TEKNISKA HÖGSKOLA

Abstract

Video cameras are the primary equipment used for indoor surveillance. There are however areas where alternatives are needed as the use of cameras is sensitive or forbidden, e.g. in homes, bathrooms or dressing rooms. A more privacy-preserving method is using a radar. The interest in radar-based surveillance indoors has increased in recent years with the development of high resolution radar sensors that are better at handling the challenges of indoor environments.

This thesis proposes a classification pipeline which aims to find people in a radar point cloud and classify their pose as either standing, sitting or lying down. Four classification models are implemented: one Random Forest Classifier, two PointNet-based classifiers of different sizes and a baseline model for comparison. These models are evaluated on realistic data from a home-like environment.

All classifiers performed better than the baseline model, with the smaller PointNet-based classifier achieving the best performance. The results show that it is feasible to use radar for simple pose classification in real-world environments.

Keywords

indoor radar, FMCW radar, point cloud, classification, Machine learning, Deep learning, PointNet

Acknowledgements

We would like to thank our industry supervisors Anders Skoog and Santhosh Nadig for all the support and guidance throughout this thesis.

We also want to thank our supervisor at Lund University, Alexandros Sopasakis for the support in technical questions and comments on the report.

Contents

Chapter 1

Introduction

1.1 Background

For indoor surveillance, video cameras are the standard equipment used to monitor rooms and areas as they provide an easy way to see what is happening. But there are situations where cameras cannot be used. Examples of this are bedrooms and bathrooms where many people don't want to be recorded by a camera. Since there are legitimate reasons to monitor these kinds of areas it would be great to have a privacy retaining surveillance method.

This is where a radar could be an alternative to cameras. A radar can output detections in the form of point clouds which represent the physical objects moving in the area. These point clouds contain less privacy-sensitive information about a person, making radar a better alternative to video in sensitive areas. Using radar for indoor surveillance has its challenges though.

Radar-based surveillance systems are mostly used outdoors as they perform well in open spaces and retain their performance in conditions where cameras often struggle. This includes rain, fog and difficult lighting conditions; radars work just as well in complete darkness as in daylight. But for the indoor use case there are some difficulties in using a radar. The biggest issue is that confined spaces usually contain many objects that can act as reflective surfaces which in turn scatters the signal and gives noisy measurements. Recently, radars with much higher resolution have been developed which could enable new ways to use radar indoors. In this thesis we will examine methods to both find and give information on people moving in an indoor environment.

1.2 Purpose

The purpose of this thesis is to create a data-processing pipeline and classification model that can follow a person in a room and classify his or her pose over time using point cloud data from a high resolution radar. The different poses that are examined are: standing, sitting or lying down. These three classes are chosen because almost all human activity can be classified into one of these poses.

The questions that this thesis seeks to answer is:

- How should point cloud data from a high resolution radar be processed to enable classification of human poses?
- Can a model using a Random Forest Classifier reliably classify the pose of a human?
- Can a model using PointNet reliably classify the pose of a human?

1.3 Prior Work

1.3.1 Object classification on high resolution radar data

Object classification on high resolution radar data is a task which has seen some activity recently, especially for automotive radar. A review article [1] lists a total of 23 articles which have attempted this task, with very different approaches and mixed results. One large difference between the articles is how the radar data is represented, which includes occupancy grids, range-doppler-azimuth maps, micro-doppler signatures and point clouds. Since most articles use their own datasets it is hard to say which representation of data is best for deep learning purposes [1]. Since this thesis deals with point clouds from a high resolution radar we will present the current state of deep learning on general point clouds and point clouds generated from radar data.

1.3.2 Deep learning on point clouds

Utilizing deep learning for object classification on point clouds is an active task with a lot of recent activity. Point clouds are unordered and irregular which makes it infeasible to directly apply computer vision techniques on them, leading to a number of different approaches on how to utilize deep learning on point clouds [2]. Review articles on the state of deep learning on point clouds [3, 4, 5, 6] categorize the architectures slightly differently. A general theme for these reviews is to separate models which operate directly on a point cloud from methods which use a grid based approach where the point cloud is converted into forms where deep learning methods from computer vision can be used.

The grid based approaches include models which first convert the point cloud into voxels, "three dimensional pixels", to then apply 3-dimensional convolutions on the voxels. One problem with voxelization is the often sparse nature of point clouds - many voxels do not contain a point [5].

Another grid based approach is the multi-view approach where a point cloud is transformed into several two-dimensional "images" which can be fed into two-dimensional Convolutional Neural Networks (CNN:s). A central problem for the multi-view approach is how to fuse together the features from each view [5].

There also exists several models which operate on raw point clouds. These include PointNet, which is considered to be one of the most influential architectures for deep learning on point clouds. PointNet differs from other most point cloud networks because it does not extract features with convolutions but with a combination of multi-layer perceptrons

and max pooling layers [6]. Since we use a modified PointNet architecture the PointNet is explained in more detail in Section 2.8.

1.3.3 Point Cloud Classification Datasets

There exists several datasets that are used for benchmarking. One of the most used dataset is ModelNet40 which contains CAD models of common objects which can be turned into point clouds by sampling the surface of the CAD model [4, 7]. This was the dataset used for training in the original PointNet paper [8]. There exists several other datasets that are used for the object classification task on point clouds such as ScanObjectNN, ScanNet, SydneyUrbanObjects and ShapeNet [3]. The point clouds in these data sets are synthetic, obtained from 3-dimensional scans of the real world using a RGB-D sensor (similar to the Microsoft Kinect) [9, 10] or obtained with a Lidar [6]. Something to note is that none of the commonly used datasets for object classification consists of radar data.

1.3.4 Object classification on radar point clouds

The more specific task of object classification on radar point cloud data can be seen in many ways as a subtask of classification of general point clouds. There are some caveats though: radar point clouds are generally more sparse than many of the point cloud datasets used for training classification models [11]. Radar point clouds also only contain points from the surface that the radar "sees", while some other point cloud datasets sample points from all sides of the object [8]. Additionally, radar point clouds often only contain the points in the scene that move. Classification with radar point clouds is therefore done on objects that naturally move, like humans, cars, bikes etc. [1].

Most articles related to classification on radar point clouds mentioned in a review on deep learning on mmWave radar by Abdu et al. [1] have automotive applications. The non-automotive articles generally focus on human activity recognition, some examples of activities that are included are boxing the air and doing squats.

In [12] the authors voxelize each point cloud, apply a 3-dimensional CNN on these voxelized frames, flatten the output from the CNN and then apply a bidirectional LSTM on this flattened output. The "prediction" from the LSTM is then fed into a 5-way softmax which gives a distribution over the 5 activities. By doing this they obtain a 92 % accuracy on their activity recognition dataset which consists of five different activities. The subjects perform the activity at a fixed distance in front of the radar.

In addition there are also some articles which attempt to recognize people based on their gait (i.e. their way of walking) [13]. In [14] the authors use a quite heavily modified version of the Pointnet architecture on radar point clouds to do this. The Pointnet architecture is also successfully used to classify radar point clouds in [11], showing that it is feasible to use Pointnet on radar data.

Chapter 2

Theory

2.1 Radar

Radar is an acronymn for Radio Detection And Ranging and is a type of sensoring device that uses radio waves to find the range, velocity and angle of targets. Radars were traditionally used for military tracking of aircrafts and vehicles. But modern radar systems have a wide range of applications, such as 2D- and 3D mapping, collision avoidance and earth resources monitoring[15, p. 3].

A radar works by sending out radio waves using transmitting antennas. The radio waves are then reflected and scattered by objects in its surroundings which is then received with a receiving antennas. In this thesis, a Frequency-Modulated Continuous-Wave radar (FMCW radar) is used.

2.1.1 FMCW Radar

The FMWC radar is a type of radar which works by sending out signal-packets called *chirps*. Each chirp consist of a sinusoidal signal that increases in frequency linearly over time. The rate of this increase is called the slope S and the range of frequencies spanned by the chirp is called the bandwidth B. We can see an example of a chirp in Figure 2.1. The reflected signal gets received and mixed with the transmitted signal which creates an intermediate frequency signal (IF-signal) as a sinusoidal signal.

A frame consist of this signal from multiple chirps and multiple antennas. The IF-signal from a frame with N chirps and K antennas can be written as[16],

$$x_{IF}(t_s, n, k) = A \cdot \sin(2\pi(f_r t_s + f_v n + f_\alpha k) + \phi_{IF}), \tag{2.1}$$

where A is an amplitude scalar, t_s is the sample time within a chirp, $1 < n < N$ is the chirp number in the frame, $0 < k < K - 1$ is the antenna number, ϕ_{IF} is the phase of the signal and f_r, f_v, f_α are different frequencies from the targets location and velocity.

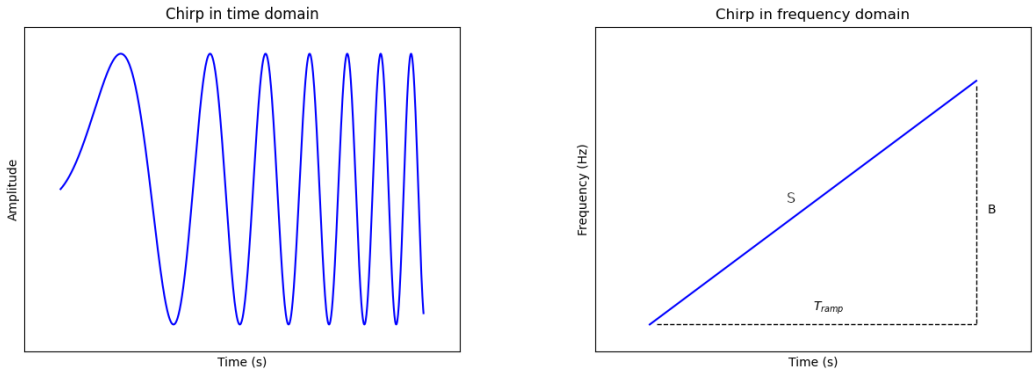


Figure 2.1: A chirp from the FMWC radar in time domain (left) and frequency domain (right). The bandwidth B is the range of frequencies the chirp sends out and the transmitting time (ramp time) is T_{ramp}. The slope S is how fast the signal increases in frequency and is given as the bandwidth divided by the ramp time.

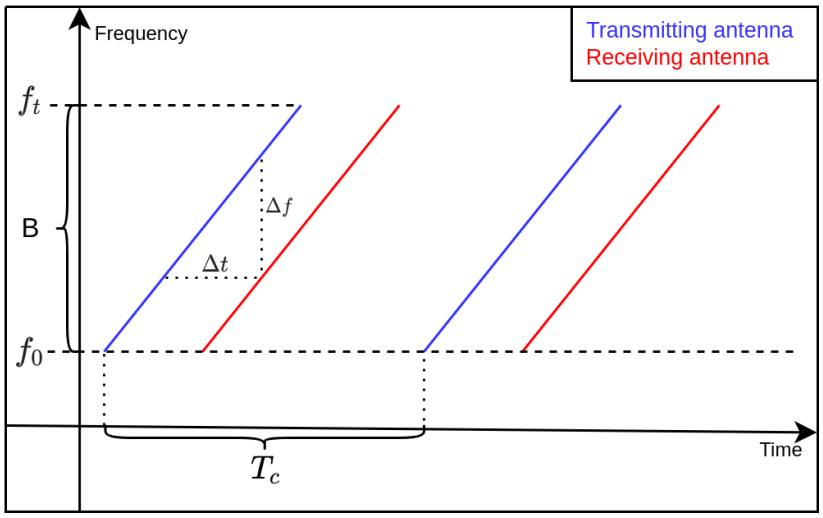


Figure 2.2: Two transmitted and received chirps from an FMCW radar with one target present in its range. The received signal is a time-delayed version of the transmitted signal and the frequency-difference will be close to constant throughout the chirp for a target.

From the IF-signal (2.1) we can calculate the following measurements for each target:

- range, r
- radial velocity, v_r
- azimuth angle, θ
- elevation angle. ϕ

2.1.2 Range of targets

If we have a target at the radial distance (range) r from the radar, it will take the signal a time Δt to travel to the target and back. This time difference is given by,

$$\Delta t = 2\,\frac{r}{c}, \tag{2.2}$$

where c is the speed of light[15, p.27]. This time difference is found as a frequency f_r in (2.1),

$$f_r = S \cdot \Delta t = S \cdot 2\frac{r}{c}, \tag{2.3}$$

where S how fast the signal increases in frequency. This is illustrated in Figure 2.2. The range is then found as,

$$r = \frac{cf_r}{2S}. \tag{2.4}$$

The smallest distance between two targets where we can resolve both targets (the resolution) is,

$$r_{res} = \frac{c}{2B}, \tag{2.5}$$

where B is the bandwidth of the signal. With a larger bandwidth we get a smaller and better resolution.

2.1.3 Velocity of targets

To detect motion we need to send multiple chirps separated in time by a constant time T_c. The target will have moved a small distance, leading to a small frequency difference f_v between two chirps in (2.1) due to the Doppler effect[15, p. 23]. For a velocity of v_r this is calculated as,

$$f_v = \frac{2v_r T_c}{\lambda}, \tag{2.6}$$

where λ is the wavelength of the signal in the center frequency. The velocity is then found in the signal as,

$$v_r = \frac{\Delta f_v \lambda}{2T_c}. \tag{2.7}$$

Since objects that doesn't move will have the same phase along all chirps we can remove all non-static objects, e.g. the background, from the signal by simply removing the mean[15].

2.1.4 Angle of arrival

To find the angle of an object compared to the direction of the radar we need at least two antennas separated by a distance d. By assuming the target to be far away we can approximate the signal to have a flat wave front as seen in Figure 2.3. If the signal arrives at an angle of α, it will have to travel different distances to reach the two antennas. This extra distance can be found as $\Delta d = d \cdot sin(\alpha)$, and will result in another frequency difference f_α between two antennas in (2.1). This frequency difference is calculated as,

$$f_\alpha = \frac{d \sin(\alpha)}{\lambda}. \tag{2.8}$$

When this frequency is found we get the angle of a target as,

$$\alpha = \arcsin\left(\frac{\lambda f_\alpha}{d}\right). \tag{2.9}$$

By using multiple antennas in both vertical and horizontal directions we can get both the azimuth angle and the elevation angle of targets.

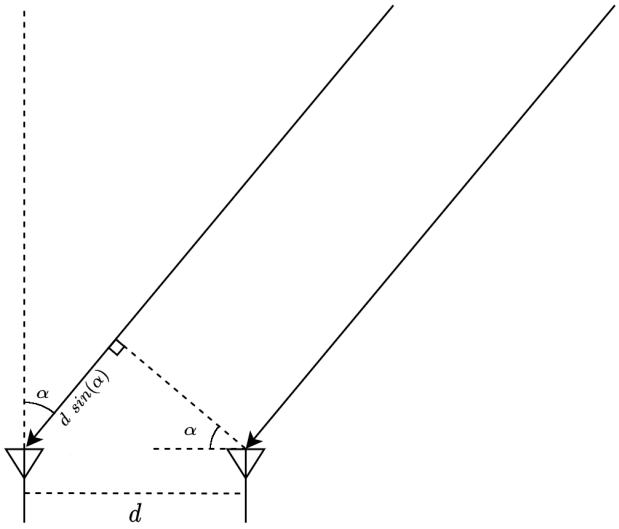


Figure 2.3: With two antennas, the angle of a target (α) will affect how far the signal has to travel to reach each antenna. This results in a frequency difference in the IF-signal between two antennas.

2.1.5 Fast Fourier Transform

Since all the information is present as different frequencies in (2.1), the data processing for the incoming radar signal needs split the signal into it's different frequency components. The Fast Fourier Transform (FFT) is a fast algorithm find the Discrete Fourier Transform (DFT) of a discrete-time signal.

For each chirp, the signal is sampled and stored into an array. The entire frame is then all of the chirps from the different antennas concatenated into a cube. An example of this cube is seen in Figure 2.4. The different directions in the cube corresponds to the different variables in (2.1) and will contain the different frequencies to find the information about the targets. To find the range of targets, an FFT is calculated in the IF-signal in sample time. The velocity of the targets can then be found by performing another FFT between chirps. Lastly, the azimuth and elevation angle can be found by an FFT between the signals of different combinations of the antennas.

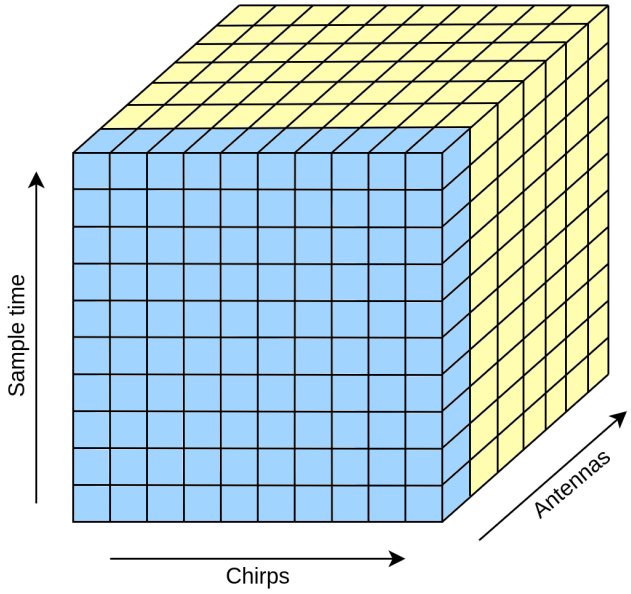


Figure 2.4: The data cube that represents a frame measured by the radar. The FFT can be used in different directions of the cube to find the targets in the area. In the sample time-direction the FFT will produce the frequencies f_r that maps to the range of a target. In the chirp-direction the velocity of target can be found with f_v. The different angles can be found in the antenna-direction by combining the signal of different antennas depending on their location in space compared to each other.

2.2 Point clouds

Point clouds are an unordered set of points in a coordinate system. They are often used as a representation of a three-dimensional object where each element in the point cloud is taken from the surface of the object, meaning that in Cartesian coordinates each point will have three values: (x, y, z) [17]. Each point can also have additional features such as color or velocity, resulting in a higher dimensional point cloud.

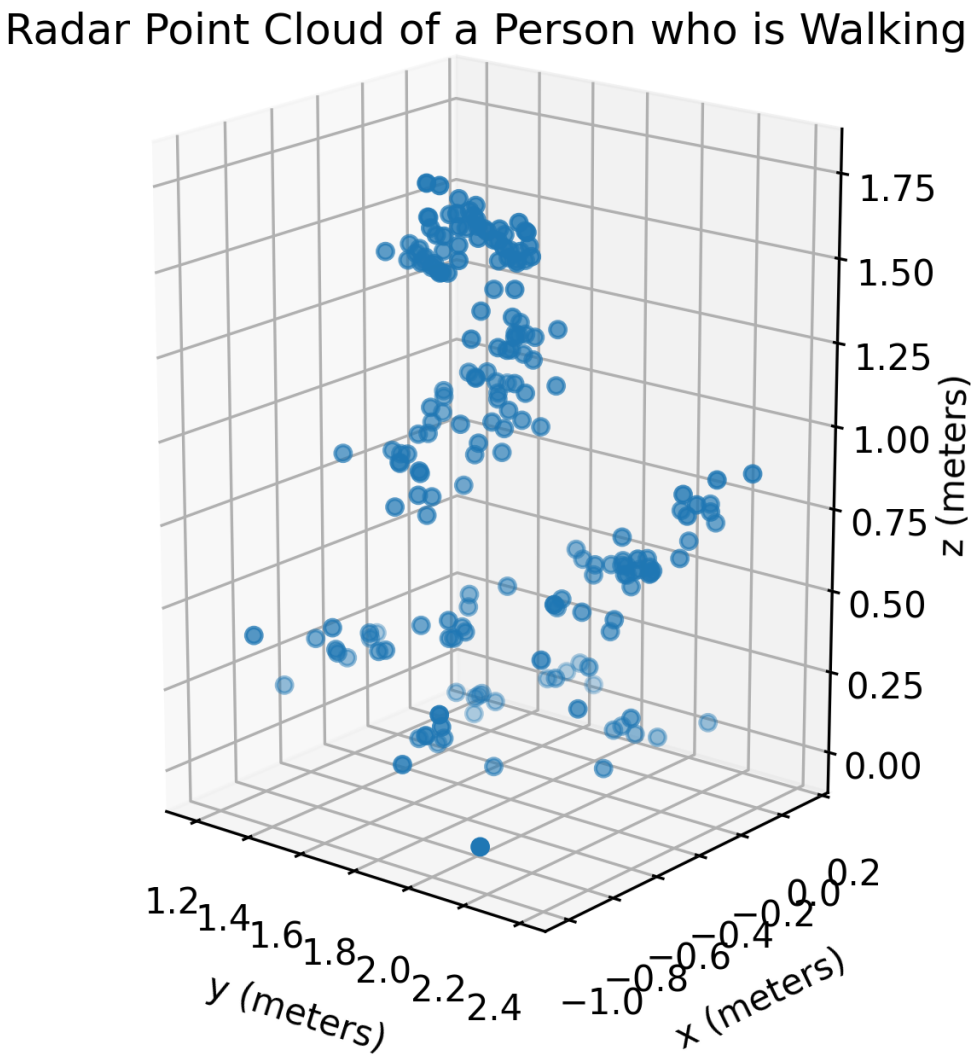


Figure 2.5: An example of a point cloud captured by a FMCW radar, depicting a person who is walking. The point cloud contains 200 points. Points further away from the point of are more transparent than closer points.

The properties of a point cloud is highly dependent on how it was captured. For example, ScanObjectNN, a dataset of point clouds captured with a depth camera has more than 1024 points per point cloud [10]. The point clouds which are used in this thesis have significantly fewer points, which can be seen in Figure 2.5.

2.2.1 Clustering

As is normal with the type of radar used in this thesis, there exists some noise in the measurements. This means that some of the points received from the radar need to be removed in order classify the point cloud well.

Points corrensponding to noise are usually less dense than the points originating from a target, so using the density of points is generally a good way to tell what is a target or not. A popular and robust method to find clusters this way is DBSCAN, which is described in Section 2.2.2 below.

2.2.2 Density-Based Spatial Clustering of Application with Noise

Density-Based Spatial Clustering of Application with Noise (DBSCAN) is an unsupervised density-based clustering algorithm[18]. The algorithm has two parameters which defines its behavior. The first parameter is `MinPoints` and describes the minimum number of samples needed in the neighborhood of a point for that point to be considered a *core point*. The second parameter is ϵ and describes the distance around a point which is considered to be its neighborhood. Any distance function can be used but most often the euclidean norm is utilized. One strength of DBSCAN is that it does not require the number of clusters to be specified before running.

Tuning the parameters for a specific task can be difficult as small changes in the parameters can change the outcome drastically. This task also becomes harder when the data varies in density in different areas. The algorithm performs better on data where clusters have similar density.

The algorithm works by looping through all points and finding *core points* that have at least `MinPoints` points in its neighborhood. When a core point is found, a new cluster expands from it. The points which are reached by a cluster but that are not core points are called *border points*. All other points are labeled as *noise*. We provide the pseudocode for DBSCAN in Algorithm 1. The $getNeighborPoints$ function finds all the neighbors of a given point. With this algorithm we get a label for each point in a point cloud. An example of the result of the algorithm can be seen in Figure 2.6.

Algorithm 1 DBSCAN [18]

```
1: function DBSCAN(Points, MinPoints, Eps)
2:     C = nextId(noise)
3:     for point in Points do
4:         if point.label == unclassified then
5:             if ExpandCluster(Points, point, C, MinPoints, Eps) then
6:                 C = nextId(C)
7:             end if
8:         end if
9:     end for
10: end function
11:
12: function ExpandCluster(Points, point, C, MinPoints, Eps) : boolean
13:     reachedPoints = getNeighborPoints(p)
14:     if reachedPoints.size < MinPoints then
15:         point.label = noise
16:         return False
17:     else
18:         SetId(reachedPoints, C)
19:         reachedPoints.delete(p)
20:         while reachedPoints is not Empty do
21:             currentPoint = getNext(reachedPoints)
22:             neighbors = getNeighborPoints(currentPoint)
23:             for newPoint in neighbors do
24:                 if newPoint.label in [unclassified, noise] then
25:                     if newPoint.label == unclassified then
26:                         reachedPoints.append(newPoint)
27:                     end if
28:                     newPoint.label = C
29:                 end if
30:             end for
31:             reachedPoints.delete(currentPoint)
32:         end while
33:         return True
34:     end if
35: end function
```

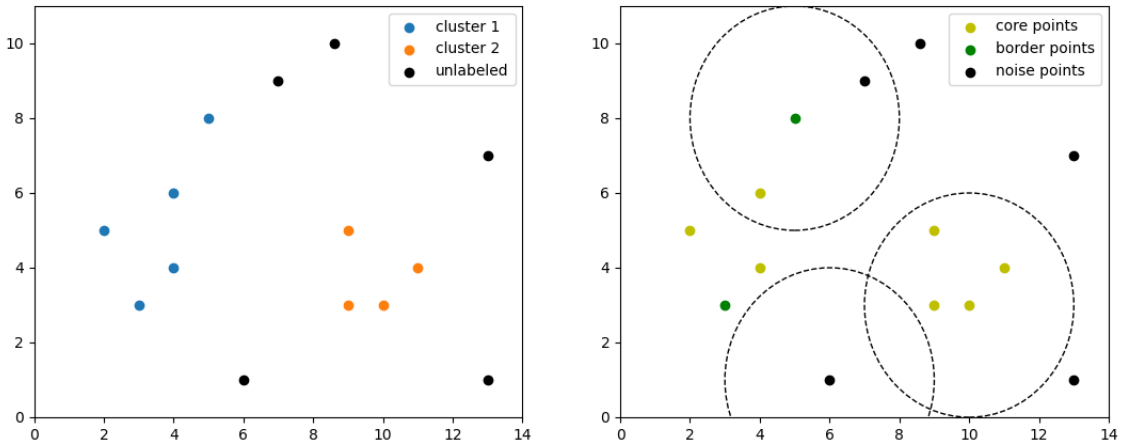


Figure 2.6: An image showing the result of a DBSCAN in 2D using the parameters `minPoints` $= 4$, $\epsilon = 3$. In the left plot we see which points are labeled as cluster and which cluster id they get. In the right plot we can see examples of core points (yellow), border points (green) and noise points (black) with a circle showing the distance ϵ around one point of each type.

2.3 Random Forest Classifier

A random forest classifier is a classifying structure/algorithm that takes an input vector $\mathbf{x}$, with N variables, and output a label y with m possible outcomes. This is done using a collection of *decision trees*.

2.3.1 Decision trees

A decision tree is a tree that divides the input space by splitting at a value in one of the input variables. A simple example of this can be found in Figure 2.7. When training a decision tree for a set of training samples, one starts with a root and no divisions. Then the input space is split into two subspaces so that the maximum amount of information is gained. This is done by looking at the possible information gain by splitting a subspace in each input variable. The information gain is calculated with either entropy or Gini impurity. Normally the Gini impurity is used and it is calculated as,

$$G(x) = 1 - \sum_{i=1}^{m} p_i(x)^2, \tag{2.10}$$

where $p_i(x)$ is the probability to pick a sample with label i from the leaf node. The total information score is then a weighted sum of the Gini impurity for each split subspace. For a split to gain information, the total Gini impurity should decrease as much as possible.

The decision tree keeps splitting each subspace into smaller subspaces until each training sample is separated into a subspace that only has samples with the same label. This would leave the Gini impurity at 0 for each subspace.

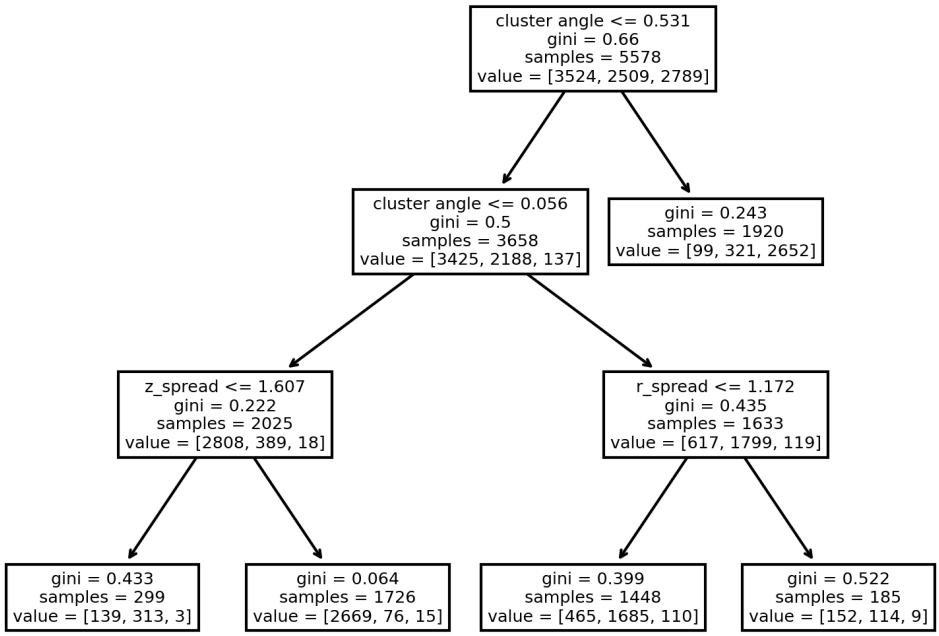


Figure 2.7: A simple example of a decision tree fitted to the data in this thesis with some constraints. The maximum depth was set to 3 and the maximum leaf nodes was set to 5. This forced the decision tree to stop splitting even if the Gini impurity is not 0 in the leaf-node. In each branch node, we see the name of the feature that the data is split by. The *value*-vector is telling how many samples of each class has ended up in the node.

2.3.2 Random forest

A random forest is a collection of decision trees that each classifies an input. The final prediction of the model is given by a majority vote from the individual decision trees. When training a random forest, each decision tree is trained on a new dataset created by bootstrapping from the original dataset. This is done by randomly picking K samples from the original set with replacement. The parameter K can be chosen to any number but usually it is picked as the size of the original dataset.

Training a random forest is a stochastic process since the trees will be different from each other. Because each tree is fitted well for the data it has seen, it will have low bias but high variance. By taking the majority vote of all trees, the variance should be reduced. This increases the bias by some amount, but generally boosts the performance of the model compared one single tree [19].

2.4 Artificial Neural Network

An artificial neural network (ANN) is a system consisting of interconnected *units* which are inspired by the neuron, a type of nerve cell in our brains. Each unit has a number of inputs and combines these to transmit one output signal, this system corresponds to synapses and the axon in biological neurons. Several units can then be connected to form a network and by adapting the strength of connections between units the network can be trained to perform tasks like handwriting recognition. This adaptation is done by *training* the network with a number of training patterns [20].

2.4.1 Multi-layered perceptron

One of the most common form of ANN is the multi-layered perception (MLP). It is a network consisting of multiple layers of nodes where each node is connected to all nodes in the neighboring layers. A computation using the network is done by cascading computations between then layers. The first layer is the input layer which consists of a vector of values that is given by the user to the network. The layers after the input layer and before the last layer are called *hidden layers*. These are the layers which output is not shown in any way during a computation. The last layer is called the *output layer* and is what the network will output. An example of this can be seen in Figure 2.8.

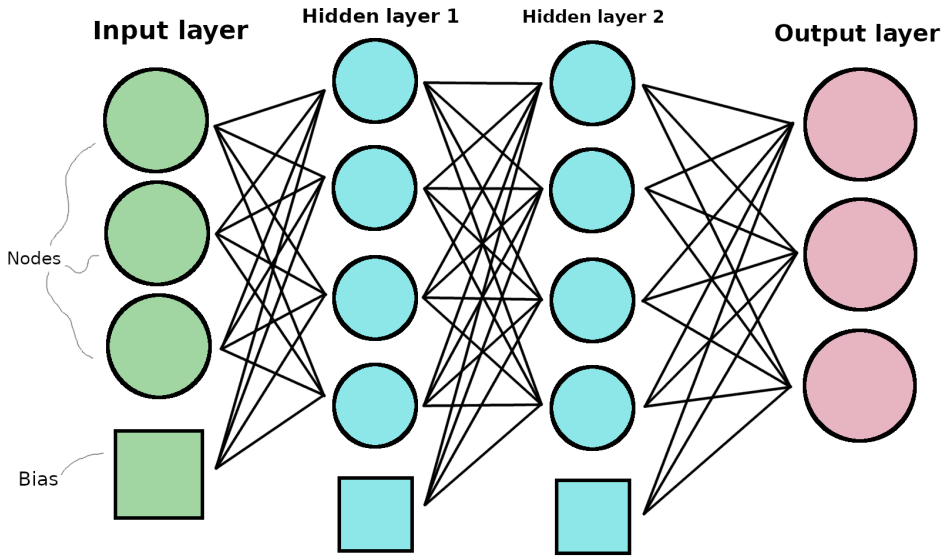


Figure 2.8: A simple neural network with 2 hidden layers. The total number of parameters in this network would be

Each node in the layer has a simple formula for calculating its output. For a node with M connecting nodes from the previous layer we get a formula for its output as,

$$a^k = f\left(\sum_{i=1}^{M} w_i a_i^{k-1} + b\right) = f\left(\mathbf{w}^T \mathbf{a}^{k-1} + b\right), \tag{2.11}$$

where w_i is the edge weight connected to each input a_i^{k-1} and b is the bias weight. The function f is a non-linear activation function. Activation functions and can be chosen in a number of ways. This is described more in Section 2.4.2.

The output of an entire layer becomes a vector. Since each node has a linear model before the activation function the entire layer can be described by a matrix equation,

$$\mathbf{a}^k = g(\mathbf{a}^{k-1}) = f\left(\mathbf{W}\mathbf{a}^{k-1} + \mathbf{b}\right), \tag{2.12}$$

where now $\mathbf{W}$ is an $N \times M$ matrix containing all the edge-weights, $\mathbf{b}$ is the bias vector. The function f is now a vectorized version of the activation function.

As each output $\mathbf{a}^k$ is the input to the next layer, the entire network can be written as a single mapping with a composition of the function for all the layers,

$$F(\mathbf{x}, \boldsymbol{\theta}) = f^L\left(\mathbf{W}^L f^{L-1}\left(\mathbf{W}^{L-1}\left(\dots f^1(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1)\dots\right) + \mathbf{b}^{L-1}\right) + \mathbf{b}^L\right), \tag{2.13}$$

expanding out to the L number of layers in the model. The variable $\boldsymbol{\theta}$ here represents all the edge weights and biases in the network that can be fitted to the data.

2.4.2 Acitvation functions

Activation functions are used in the layers of an Neural Network to add non-linearity to the model that is otherwise just linear. Since the networks are often used to imitate non-linear mappings, the addition of activation functions are important for good performance.

In this thesis, the ReLU-function is used for all layers except the output layer. The formula for the ReLU-function is,

$$\text{ReLu}(x) = \max(x, 0), \tag{2.14}$$

which only activates the node if it has a positive value. This is the default activation function recommended to use with most feedforward neural networks as it creates a non-linear function while still retaining the properties of linear models that make them easy to optimize [21, p. 170].

For the output layer, the softmax-function is used. This function transforms the values in the output-layer to a normalized version of itself. This normalized version represent the probability distribution over the different classes [21, p.180-181]. The formula for the softmax-function is given by,

$$\text{Softmax}(\mathbf{x})_k = \frac{e^{x_k}}{\sum_i e^{x_i}}. \tag{2.15}$$

2.4.3 Optimization

When training or fitting the neural network to the problem, one normally uses a loss function for optimization of weights connected with the different inputs $\mathbf{x}_i$ and their outputs $\mathbf{y}_i$. We describe the loss function as a mapping,

$$L : \mathbf{X} \times \mathbf{Y} \times \boldsymbol{\theta} \to \mathbb{R}, \tag{2.16}$$

such that when we minimize the loss function, we also make the network output closer to the desired output. By doing this, we can see this entire task of fitting the network to the training data as an unconstrained optimization problem. The loss function used in this thesis is the *categorical cross-entropy*-function,

$$L(\mathbf{X}, \mathbf{Y}, \boldsymbol{\theta}) = -\sum_{j=1}^{N}\sum_{i=1}^{K} \mathbf{Y}_i^j \ \log(F(\mathbf{X}^j, \boldsymbol{\theta})_i), \tag{2.17}$$

where we have K classes, N training samples and $F(\mathbf{X}^j, \boldsymbol{\theta})$ is the neural network described as (2.13).

When training the network, you start with the partial derivative of the loss function L with respect to the output of the network and then use backpropagation [22, p. 241] to find the gradient with respect to all parameters in $\boldsymbol{\theta}$. A gradient based opt imization algorithm can then be used to find the parameters that solves (2.16). Some regular algorithms used for this are Stochastic gradient descent (with and without momentum) [21, p.290], and the Adam method[23].

2.4.4 Regularizers

A common problem for large networks is that they optimize too well to the data that is used for training which does not create a good model for the general problem.

This is a case of overfitting and a model is prone to this when the number of parameters in the model are large. But finding the right amount of parameters in different parts of the model would be a hard task to do manually. But the complexity of the model can instead be controlled by the addition of a *regularization term* [22, p. 256].

The regularization term is a penalizing term that gets added to the loss function. The penalizing term is there to limit how large the different weights in the model can be, as large weights indicates that model relies a lot on single parameters in the model. In this thesis, the L2-norm is used as a regularization term. By adding this, the loss-function (2.16) becomes,

$$L(\mathbf{X}, \mathbf{Y}, \boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_2^2. \tag{2.18}$$

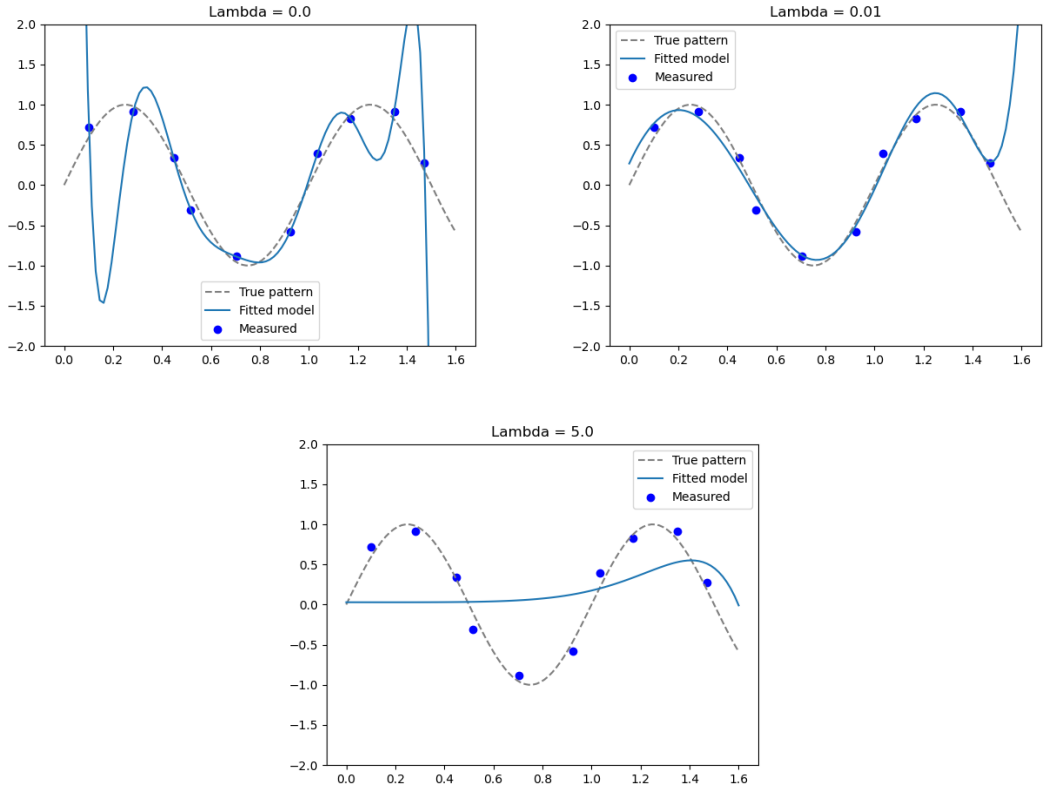


Figure 2.9: Fitting a 10th degree polynomial to a sinusoidal curve with noise using different regularization scalars λ. We can see in the case where we have $\lambda = 0$ (upper left plot) the polynomial fits perfectly to the measured points and the polynomial does not capture the underlying model. If instead a more suitable $\lambda = 0.5$ is used (upper right plot) we see that the polynomial follows the underlying pattern well in the interval where the measured points are. A too large $\lambda = 5$ (lower plot) will instead limit the weights so much that the polynomial can not follow the measurements.

where λ is a scalar chosen for a good balance between keeping the weights small and matching the output to the labeled data. It is important to choose λ well, as a small penalizing term will not affect the model enough and a too large penalizing term will limit the model. We see an example of this in Figure 2.9.

This way, the adjustment of the weights isn't hard limited but instead it becomes a part of the optimization function so that the same optimization algorithms can be used with the desired result.

2.4.5 Dropout

Dropout is another way to regularize a network and stop the network from overfitting. The idea of dropout is to pick a "thinned out" version of the network when training on different samples to not let parts of the network co-adapt. This should help create a model

that generalizes better[24].

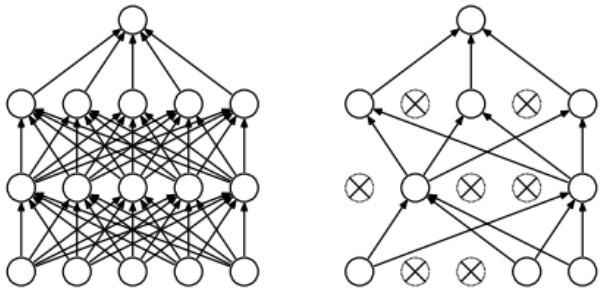


Figure 2.10: An example of a network before (left) and after (right) applying dropout. Image from [24].

During training on a mini-batch, a separate dropout will be done for each training sample. An example of this is seen in Figure 2.10. The layers will randomly drop nodes with probability p. The dropped nodes will not have an effect when inferring nor when calculating the gradients in the backpropagation. To get the weights-updates, the gradient for the nodes are averaged by number of times they got used to create the update for the batch [24].

2.5 Convolutional Neural Network

A Convolutional Neural Network (CNN) is another type of neural network which provides great performance in practical applications such as computer vision. This type of network needs order between all inputs to function, meaning that this type of network would not work directly on the unordered point cloud. We will here give a quick overview on how it is used on images.

When talking about CNN:s the term "convolution" is often used to also denote the operation cross-correlation, which is in practice very similar to the convolution-operation. We will here use the terms interchangeably. Here, K is the kernel (also called a filter) of the convolution and I is an image on which it is applied on. In practice, the dimensions of K is small than those of I. Since I and K are finite and discrete the convolution operation can be written as,

$$S(i,j) = (K * I)(i,j) = \sum_n \sum_m I(i+n, j+m)K(m,n), \tag{2.19}$$

for two dimensions. Graphically this can be seen as "sliding" the kernel across the image I and performing elementwise multiplication of the values which are in the intersection of I and K and adding these values together. The result of this operation will be a new image S with dimensions $((i_1 - k_1 + 1) \times (i_2 - k_2 + 1))$ (so called "valid" convolution). This can also extend further into more dimensions.

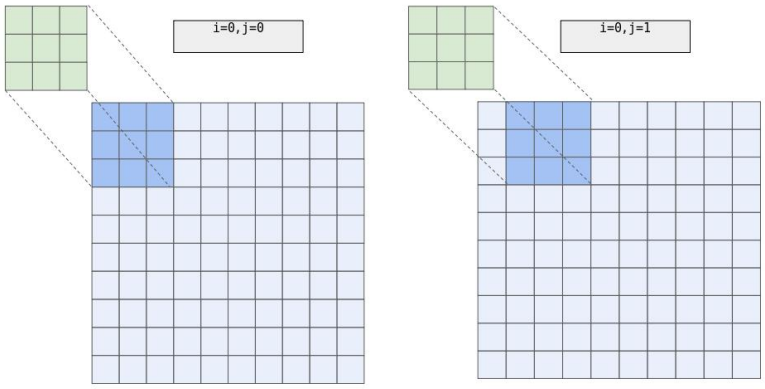


Figure 2.11: Visualization of a 3×3 kernel sliding over a 10×10 image.

In classical computer vision the kernel can be hand-crafted to perform operations like edge detection but in the domain of deep learning the kernel weights are instead learned to perform a specific task.

2.5.1 Convolutional layer

A convolutional layer is a layer that takes an image and performs convolutions on it with multiple kernels of a set size. Each output pixel is then run through a non-linear activation function (see Section 2.4.1) and then an optional pooling layer which downsamples the image by combining pixels which are close to eachother with the help of a pooling function (such as the maximum function). The output of the convolutional layer will therefore be a lower resolution image for each kernel in the layer.

CNN:s generally consists of several convolutional layers stacked on top of each other. The original inspiration for this design comes from models of the visual cortex. In those models the *receptive field* of a neuron is the size of the visual input that affects whether the neuron activates or not. This is analogous to the size of the kernel as it slides over the image [25, p. 1397-1401].

One of the main advantages of convolutional layers over fully-connected layers is that the number of weights in a convolutional layer is much lower. If a picture has size 1000×1000 a fully connected layer would have $1000 \times 1000 = 10^6$ weights (excluding biases) while a convolutional layer with a 10 kernels of size 10 would just have $(10 \times 10) \times 10 = 1000$ weights.

2.6 Dimensionality reduction

When analyzing a model or network, the different layers can become quite large in number of parameters and dimensions. The classification part of a model usually takes some feature vector as input to perform the classification. This feature vector could be computed manually or be the result of a previous part of a network. To estimate how well

classifications can be done, the data from the different classes should be separated in this feature space for the model to be able to predict with high precision.

To get an intuitive understanding of the feature space, we want to visualize it in 2 or 3 dimensions. To do this we need to reduce the number of dimensions from the feature space in some way. There are many methods to achieve this. A widely used method to reduce dimensions is the Principal Component Analysis (PCA). But this method can overlook a lot of information when the dimensions grow large. This is why this thesis uses t-distributed stochastic neighbor embedding instead for a non-linear way of conserving as much of that information as possible.

2.6.1 T-distributed stochastic neighbor embedding

T-distributed stochastic neighbor embedding (t-SNE) is a non-linear dimensionality reduction used to present high-dimensional data in lower dimensions while preserving as much of the original structure as possible. This is achieved by transforming similarity in distance to a probability. In this way, the points that are similar in high dimensions will be mapped close to each other in the reduced space in a way linear methods can not. This differs from linear methods, e.g. principal component analysis (PCA). PCA instead find the subspace spanned by the directions of most variance in the data and then projects the points onto that subspace. One problem with PCA is that it could lose a lot of information since there are many dimensions that are simply ignored.

To start creating the t-SNE mapping, the distances between high-dimensional points (x_i, x_j) are transformed into probabilities,

$$p_{i|j} = \frac{e^{-\|x_i - x_j\|^2/2\sigma_i^2}}{\sum_{k \neq i} e^{-\|x_i - x_k\|^2/2\sigma_i^2}}, \tag{2.20}$$

where σ_i is the variance of the Gaussian centered around x_i. Each of the high-dimensional points x_i gets mapped to a corresponding low-dimensional point y_i. The low-dimensional points also get a similar probability distribution,

$$q_{i|j} = \frac{e^{-\|y_i - y_j\|^2}}{\sum_{k \neq i} e^{-\|y_i - y_k\|^2}}. \tag{2.21}$$

If the distribution P is similar to Q, then these probabilities should be the same. A measure of similarity in distributions is the Kullback-Leiber divergence. A good mapping from P to Q is found by minimizing this KL-divergence over all samples as [26].

$$C = \sum_i KL(P_i \| Q_i) = \sum_i \sum_j p_{i|j} \log(\frac{q_{i|j}}{p_{i|j}}). \tag{2.22}$$

One thing to note about t-SNE is that the problem of fitting the distribution Q is non-convex, so different mapping of the same data could potentially look different. This is

not the case for PCA, as the directions of most variance is always the same in the dataset.

2.7 Hidden Markov Models

A Hidden Markov Model is a type of discrete-time Markov Chain, so we begin by defining what that is.

Discrete-time Markov Chains

A stochastic process is a sequence of random variables $X_0, X_1, X_2, ...$ where X_n denotes the state of the process at time n and X_0 is the initial state. Since $n \in \mathbb{N}$ the process is discrete in time. The collection of all state values is called the state space and is denoted by $\mathcal{S}$ [27].

We will assume that the random variables which make up the stochastic process take on values in a finite subset of $\mathbb{Z}$. The stochastic process will therefore be discrete in both time and state.

A discrete-time Markov Chain is a stochastic process which is discrete in time with the Markov property,

$$P(X_{n+1} = j \mid X_n = i_n, ..., X_0 = i_0) = P(X_{n+1} = j \mid X_n = i_n), \tag{2.23}$$

where $j, i_n, i_{n-1}, ..., i_0$ are states. The Markov property (2.23) can be described in words as "the next state is independent of all previous states given the current state" [27].

In this report we will assume that a Markov chain's state space is finite (and discrete). The term "Markov Chain" will from here on refer to a discrete time stochastic process with a finite state space. If we further assume that the probability of transitioning from state i to state j is independent of the time n we can describe a Markov Chain using a $|\mathcal{S}| \times |\mathcal{S}|$ transition matrix $\mathbf{T}$ where $\mathbf{T}_{ij}$ denotes the probability of going from state i to j.

With a Markov Chain one can model the pose of a person moving. We introduce three states: standing, sitting and lying down. We assume that Markov property is true for this process, i.e. that the next pose is independent of the previous poses given the current pose. We also assume that the probability of transitioning between two poses is some constant a, which means that the probability of staying in the same pose is $1 - 2a$. The process can then be visualized with Figure 2.12 below.

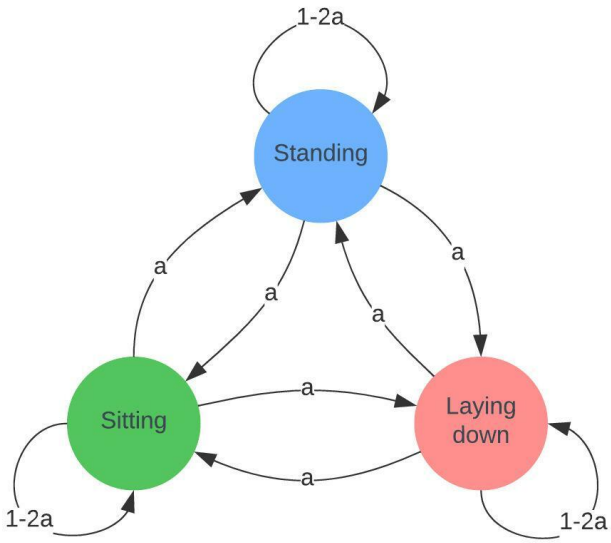


Figure 2.12: Example of a Markov Chain

The transition matrix will be,

$$\begin{pmatrix} 1-2a & a & a \\ a & 1-2a & a \\ a & a & 1-2a \end{pmatrix}.$$

Hidden Markov Models

A Hidden Markov Model (HMM) extends the idea of a Markov Chain to a setting where observations of a Markov Chain are made with a noisy sensor. By combining information about the sensor and the underlying Markov Chain it is possible to create predictions which are more accurate compared to only relying on the sensor. This task is called filtering and can be expressed computing the distribution $\mathbf{P}(\mathbf{X}_t|e_{1:t})$ with $e_{1:t}$ being the sensor readings from the beginning of the measurement to time t. This whole sequence will be known at time t.

The first component of a HMM is the transition model which refers to the transition matrix $\mathbf{T}$ of the Markov Chain. The next component is the sensor model which describes the properties of the sensor. It consists of $P(e_t|X_t = i)$ for each state i. The sensor reading e_t can be a real-valued multidimensional reading in the context of HMMs but in this thesis it will just take on the same values as X_t (the state of the Markov Model) which is what we want to observe. With the transition model and sensor model we can define the forward equation,

$$\mathbf{P}(\mathbf{X}_t|e_{1:t}) = \alpha\mathbf{P}(e_t|\mathbf{X}_t)\sum_{x_t}\mathbf{P}(\mathbf{X}_t|X_{t-1} = s)P(X_{t-1} = s|e_{1:t-1}). \tag{2.24}$$

Here, $\mathbf{P}(e_t|\mathbf{X}_t)$ is given by the sensor model and is a vector with the same number of elements as the number of states. $\mathbf{P}(\mathbf{X}_t|X_{t-1} = s)$ corresponds to a row in the transition matrix. The last expression, $P(X_{t-1}|e_{1:t-1})$, is called the forward message and can be calculated through recursion as it is the left hand side of (2.24) at time $t-1$. The term α is used for making the probabilities of the forward message sum up to one. By setting $\mathbf{f}_{1:t} = \mathbf{P}(\mathbf{X}_t|e_{1:t})$ we can write,

$$\mathbf{f}_{1:t} = \text{Forward}(\mathbf{f}_{1:t-1}, e_t),$$

where Forward updates the forward message as described in (2.24).

When we know the observation e_t at time t we take the sensor model values and place them into a diagonal observation matrix $\mathbf{O}_t$. The ith diagonal is set to $P(e_t|X_t = i)$ and the matrix will be of dimension $|\mathcal{S}| \times |\mathcal{S}|$. We can then rewrite the forward equation (2.24) as

$$\mathbf{f}_{1:t} = \alpha \mathbf{O}_t \mathbf{T}^\top \mathbf{f}_{1:t-1} \tag{2.25}$$

where $\mathbf{T}$ is the transition matrix. With this updating rule it is possible to efficiently update the forward message $\mathbf{P}(\mathbf{X}_t|e_{1:t})$ as new evidence e_t arrives [25].

2.8 PointNet

PointNet is a type of Artificial Neural Network that can do classification and segmentation on unordered point clouds. When the PointNet article was published it differed drastically from previous methods approaches of deep learning on point clouds [3]. Some other methods include quantifying the world into voxels and using a 3D-CNN network or projecting the point cloud down into different images and combining the result of different CNNs [6]. One advantage of PointNet compared to these other methods is that little pre-processing is needed as the network processes raw point clouds.

The underlying idea of PointNet is to distill a point cloud into a set of informative points together with an encoding on why these points are important. The classification of a point cloud is then only based on these informative points. One important thing to note is that PointNet bases the classification on *global* features, it does not take local structures into consideration, as is the case with CNN:s.

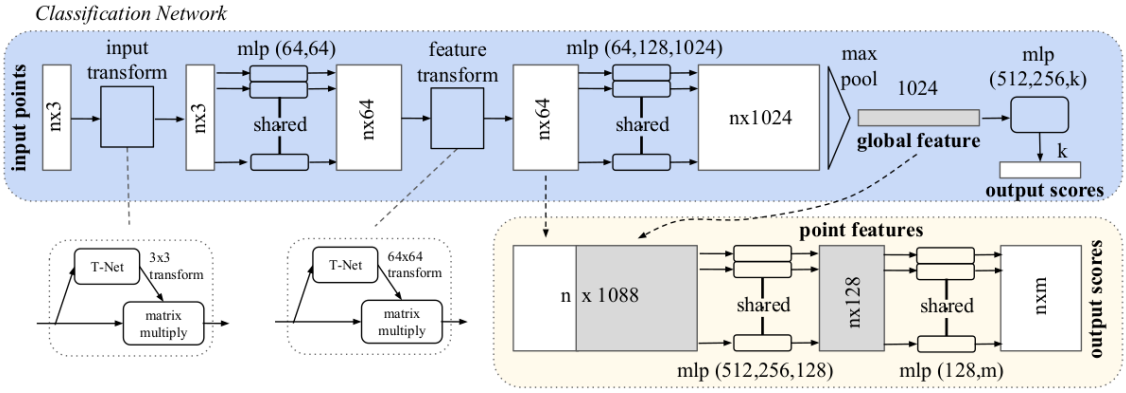


Figure 2.13: The architecture of the PointNet-network with number of parameters in each layer following the original article. Only the blue part labeled *Classification Network* is of interest in this thesis as we use PointNet for classifying targets. Image taken from the original paper[8].

One defining feature of Pointnet is the fact that the order of the input does not matter, i.e. two point clouds which contain the same points but in a different order will return the same results. This is achieved using a combination of *global max pooling* and *shared MLP:s*. We will now explain the general structure of Pointnet and how a point cloud results in a classification.

A point cloud is at first represented by a $n \times 3$ matrix, where n is the number of points in the cloud and where 3 represents the spatial dimensions (x, y, z). Each point is then expanded into a 1024 dimensional feature vector through a number of MLP:s. This sequence of MLP:s which map a 3-dimensional point into $\mathbb{R}^{1024}$ has the same weights for each point, i.e. each point is fed into the same function. This is where the term *shared-MLP* comes from, each point shares the same MLP:s.

This results in a $n \times 1024$ matrix on which a column-wise max-pooling is done, resulting in the 1024-dimensional feature vector which encodes the point cloud. Since the max-pooling is done over the points, only informative points contribute to the global feature vector. This global feature vector is then run through three fully connected layer to produce the final classification predictions.

T-Net

A part of PointNet which we have not explained yet are the so called T-nets. The T-Net is a sub-network which is added to the PointNet architecture to transform the point cloud to a "canonical" alignment. This is motivated by saying that a point cloud should give the same classification before and after it undergoes a rotation or mirroring (or a combination of these) [8]. The layer creates a specific transformation matrix A that is suited to the input sample and then multiplies this matrix with the input. Since this matrix should represent a rotation or mirroring, a regularization term aiming to make A orthogonal is introduced (all rotations and mirrorings are represented by orthogonal matrices). This is done by adding,

$$L_r = \|I - A^T A\|_F^2, \tag{2.26}$$

to the loss function, where $\|\cdot\|_F$ is the Frobenius-norm.

Although the idea of Pointnet is input-size invariant, implementations often work on a fixed number of points, i.e. n is fixed. This leads to a problem since the point clouds that are obtained from a radar vary in size. There has been some discussion on how to train and run inference on point clouds of varying sizes but in this thesis we use the fixed version of Pointnet. Another important point is that Pointnet is not limited to representing each point by spatial coordinates, it can be expanded to work with an arbitrary number of point-wise features.

Chapter 3

Data

3.1 Radar setup

The data is collected using an FMCW radar mounted on a stand on a height of 2.4 meters with a downward angle of 30°. Next to the radar we also have a camera which provides video recording of what happens in the scene. The video clips are then used when annotating the radar data. The specifications of the radar can be found in Table 3.1 below.

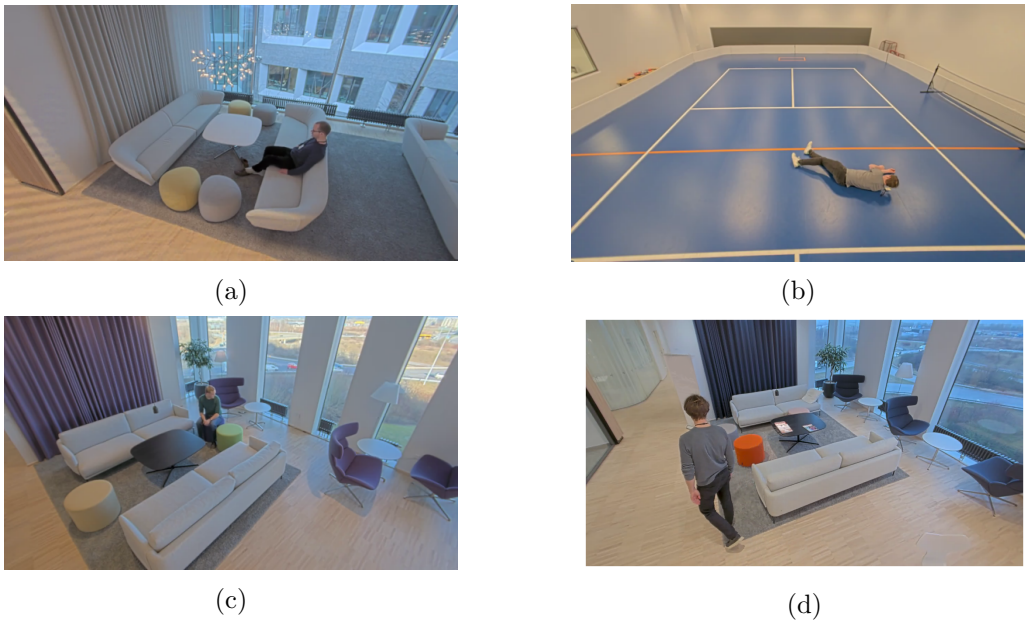
Parameter name	Value
Frame rate	10 FPS
Range resolution	0.058 m
Maximum range	18.085 m
Velocity resolution	0.0343 m/s
Maximum velocity	7.853 m/s
Azimuth angle accuracy	1°
Elevation angle accuracy	2°

Table 3.1: The specifications and settings of the radar used to collect data.

The output from the radar is a point cloud with a number of features per point such as x, y, z positions (relative to the radar), radial velocity $\dot{r}$ and signal-to-noise ratio. These features are then used to filter out points.

3.2 Recordings

In total 31 recordings are done with a single subject in each scene. The subjects are instructed to walk around the room and to sit or lay down occasionally but without any set recording schedule. The average recording is around a minute in length. A number of different recording environments are used, some of them are shown in Figure 3.1.



(a) (b) (c) (d)

Figure 3.1: Four different recording environments, images taken from the point of view of the radar.

3.2.1 Data sets

To properly train and evaluate the models the recorded data is split into two sets, a training set and a test set. This is done to ensure that the models are both trained and evaluated in a fair way.

Training set

The first set is the training set. This set contains data from 26 of the recordings from location (b), (c) and (d) in Figure 3.1 and contains a total of 11476 samples. This set was be used to train and validate the classification models. The set was then further split into training data (8695) and validation data (3038) as seen in Table 3.2.

Training	—	Validation	—
Standing	3508	Standing	1129
Sitting	2450	Sitting	961
Lying down	2737	Lying down	948
Total	8695	Total	3038

Table 3.2: A table with the division of samples in the Training set. The set itself is split into training and validation parts where the model is fitted to the training part and hyperparameters are determined using the validation part.

3.2.2 Test set

The remaining 5 recordings are used to evaluate the performance of the models. To make the evaluation robust, the location and people are different from those in the training set. The test set was recorded in location (a) in Figure 3.1.

The set itself contains 3500 samples and the division of classes is found in Table 3.3 below.

Test	—
Standing	1626
Sitting	1124
Lying down	750
Total	3500

Table 3.3: A table with the division of samples in the Test set.

3.3 Description of Point Clouds

The number of points vary quite a lot during a recording. This is due to factors such as distance between the person and the radar and how much the person moves. To describe the data we randomly pick a recording and show some statistics for it. The recording is of a person walking around and standing still in a lounge area, pictured in Figure 3.1d, for 69 seconds leading to 690 unique point clouds. In Figure 3.2 is a histogram of the number of points in the point cloud from this recording before any filtering is done.

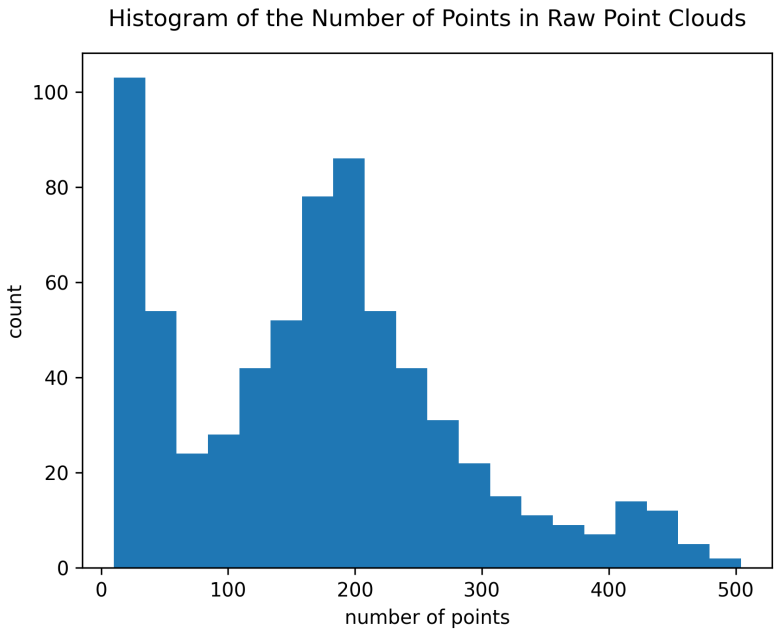


Figure 3.2: Histogram over the number of points in unfiltered point clouds over one recording. The recording consists of 690 frames, i.e. the histogram represents 690 point clouds.

As shown in Figure 3.2 there are some frames with very few points. This corresponds to frames with the subject in the recording not moving. The frames with many points have the subject walking away from the radar while still being close to it. Our hypothesis is that their back becomes an excellent surface for the radar signal to reflect off of. To make the point clouds more similar in size before classifying them a lot of data processing is done, see Section 4.2.

3.4 Annotation

To use supervised learning models annotated data is required. In the ideal case, we would like to train our whole pipeline to the task of finding the points in a point cloud which correspond to a human and classify his/her pose. Training a model for this task requires good data where every point should have an instance associated with it. Performing this kind of annotation on three dimensional point clouds is very time consuming, so this approach was deemed infeasible for the given time.

Instead we use our data processing-pipeline to produce a cluster from each raw point cloud. This clustered point cloud is then labeled as either standing, sitting or lying down. Since we only record data with a single subject in the scene we get one label per frame. Given that a person's pose usually stays the same over several seconds, multiple frames can be labeled together.

There are some potential flaws with this approach of annotation. Some frames might

be clustered badly which would then add bad samples to the data sets. But since these bad clusters could show up in real data it might be beneficial to also include them in the training data.

3.5 Data augmentation

Since the data sets are rather small compared to data sets normally used for deep learning, data augmentation can be used to synthetically create more variance in the training set. There are several ways to augment point clouds, but in this thesis they were jittered.

Jittering is defined as adding noise to each point. In our case each point is mapped as $(x, y, z, v_r) \rightarrow (x + \epsilon_x, y + \epsilon_y, z + \epsilon_z, v_r + \epsilon_v)$ where the different ϵ are noise drawn from a uniform distribution. The size of the uniform distributions for these noise terms were set to double the resolution in range for the spatial coordinates and double the resolution in velocity for v_r. We therefore get $\epsilon_x, \epsilon_y, \epsilon_z \sim Uniform(-0.1, 0.1)$ and $\epsilon_v \sim Uniform(-0.06, 0.06)$. The values for the resolutions can be found in Table 3.1.

3.6 Quality of data

For the majority of the data captured, the point cloud seems to match what is seen in the video reference. But there were some frames and parts of the recordings where the point cloud of the person did not appear as expected. This could either be due to the data processing or limitations of the radar.

One kind of problematic sample was to occlusion of body parts. This seem to happen mainly for two reasons. Either that part of the person didn't move enough to be captured or it was covered by an object in the scene. Common examples of this was when a person did not move their feet while standing or lying still but was actively doing something with their arms. Another example of occlusion can be seen in Figure 3.3 where the lower body of the person is blocked by a table.

Another problem was when the point cloud from a person was split into two different clusters. We see an example of this in Figure 3.4. Since we only record with one person in the scene a simple solution would be to just concatenate all nearby clusters. This was not possible due to noise and reflections in the room. Instead the largest cluster found was deemed as the target, which meant that only part of the body was saved as the sample. This can be found in both the training and test data, which hopefully meant that the model could learn to classify these samples correctly anyways.

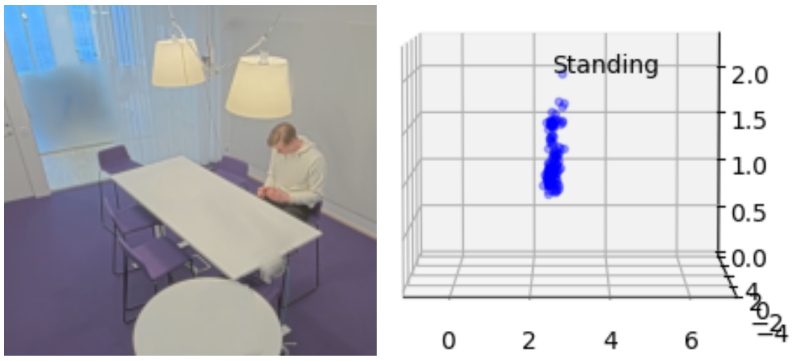


Figure 3.3: An example where part of a person becomes occluded by an object in the room. The legs of the person are covered by the table and the radar only captures the upper body as seen in the point cloud to the right. This type of occlusion of body parts can also happen if a person only moves part of their body.

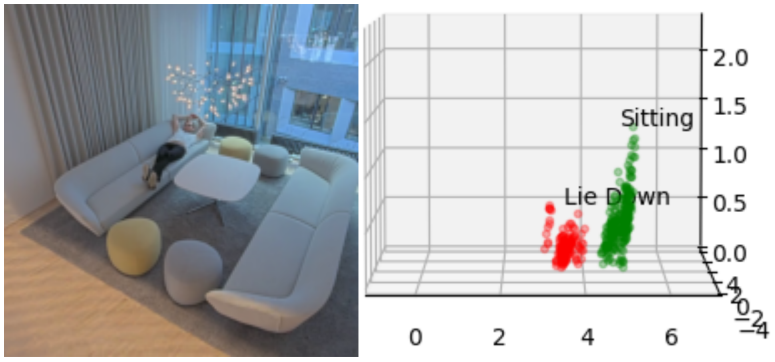


Figure 3.4: An example from the test set where the points from a person become clustered into two separate sets as a part of the body was not moving enough. The result of this is that only the largest of the two clusters gets saved in the sample which could lead to the model learning on incorrect point clouds.

Chapter 4

Methods

We will now discuss the methods used to answer the questions posed in this thesis, namely

- How should point cloud data from a high resolution radar be processed to enable classification of human poses?
- Can a model using a Random Forest Classifier reliably classify the pose of a human?
- Can a model using PointNet reliably classify the pose of a human?

In the first part of the methods section we present how we have implemented the data processing and models mentioned in these questions. After this, in Section 4.7, we describe how the implementations were evaluated, i.e. how the research questions were answered.

4.1 Overview of Implementation

In Figure 4.1 a simplified example of how the classification pipeline proposed in this thesis is shown. The scene shown in the picture is captured into a point cloud which is then filtered, clustered and classified.

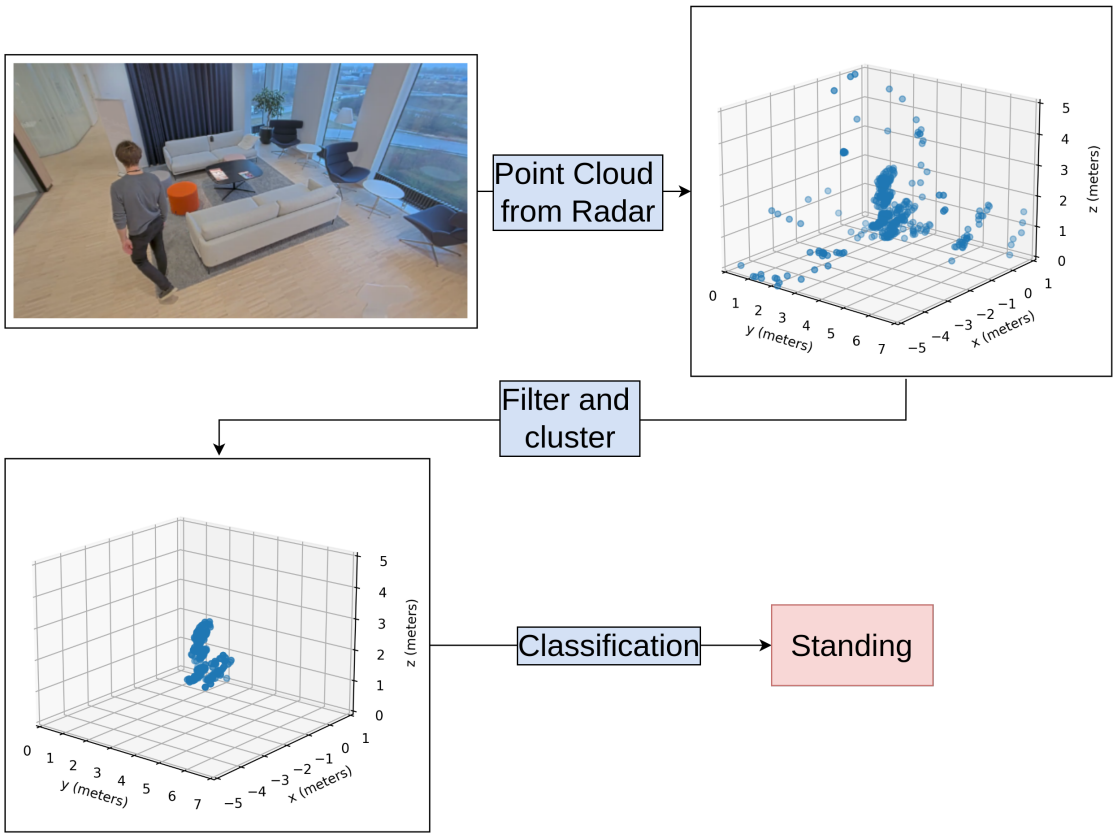


Figure 4.1: Simplified example of how the processing pipeline functions. The radar captures a point cloud of the scene shown in the picture. The point cloud is then filtered and clustered and finally run through a classifier which in this case classifies the point cloud as "standing".

A more detailed visualization of the pipeline can be seen in Figure 4.2. In addition to showing a more granular view of the pipeline this diagram shows what happens in the case that two clusters are detected in the point cloud. After the clustering step two tracks called "Track 1" and "Track 2" are created from the clusters. The name "Track" stems from the fact that the position and pose of the clusters are *tracked* over time. Each of these tracks are then classified separately.

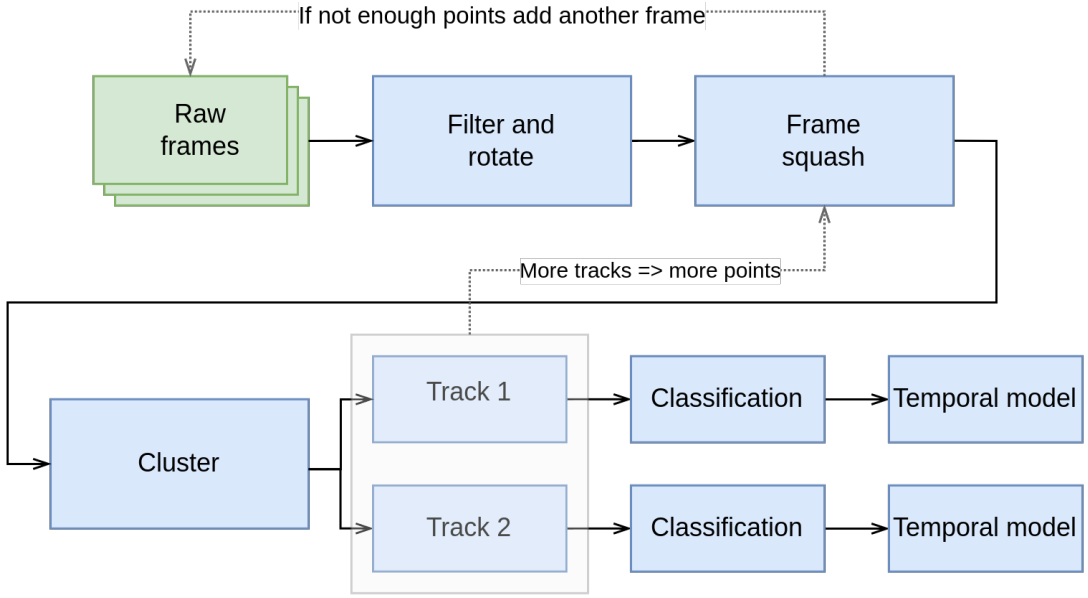


Figure 4.2: Diagram showing the processing pipeline in the case that the tracker finds two tracks. Each of the blocks are further explained in the methods section.

We will now explain these different modules and the methods used in them.

4.2 Data processing

First we will look at the data processing part of the pipeline, i.e. the blocks labeled as "Filter and rotate" and "Frame squash" in Figure 4.2.

4.2.1 Rotation and translation

The raw point cloud which the radar outputs has a coordinate system which is centered around the radar sensor. To make it easier for humans to understand the point clouds each raw point cloud is first rotated down by 30° around the x' axis and translated so that the floor is at $z = 0$. This way every point is transformed from (x', y', z') coordinates to (x, y, z) see Figure 4.3.

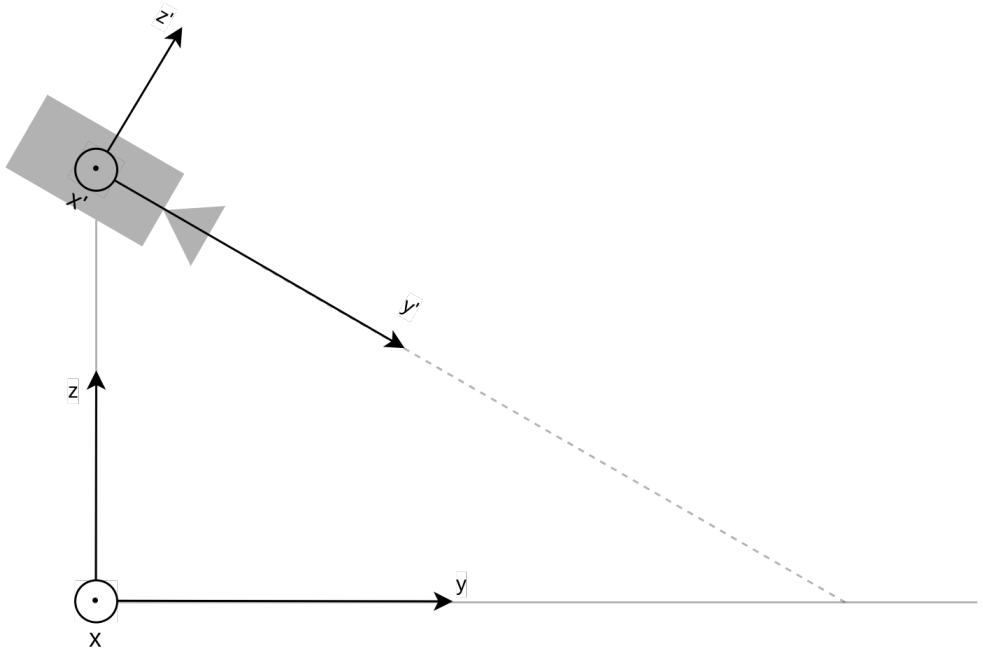


Figure 4.3: Illustration of the two coordinate systems. Each point is transitioned from (x', y', z') coordinates to (x, y, z)

4.2.2 Filtering points

Before clustering we want to remove as many non-target points as possible. We assume that targets of interest will produce a high energy signal compared to the background noise. Therefore, we remove all points with a signal-to-noise-ratio of less than $17.5 - r$ dB, where r is the radial distance to the radar. This filtration method is chosen to remove noise while still retaining points that are further away from the radar. This was chosen to keep the number of points per target similar independent of distance from the radar.

Some other physical boundaries were assumed for further filtering of noise. All targets within 0.1m from the radar were assumed to be noise and removed. The targets were also assumed to be above the floor and below 2.2m. Points outside of this range were removed.

Mapping walls in the room

One common problem with radars is the multipath phenomenon. When performing tracking and classification with a radar it is assumed that the radar signal travels from the radar to the tracked object and back, without hitting anything else on its way. This does not always hold true as the radar signal can be reflected by something else both before

and after hitting the object. In Figure 4.4 we can see what happens when a radar signal first hits a target and then a wall, leading to a track inside the wall.

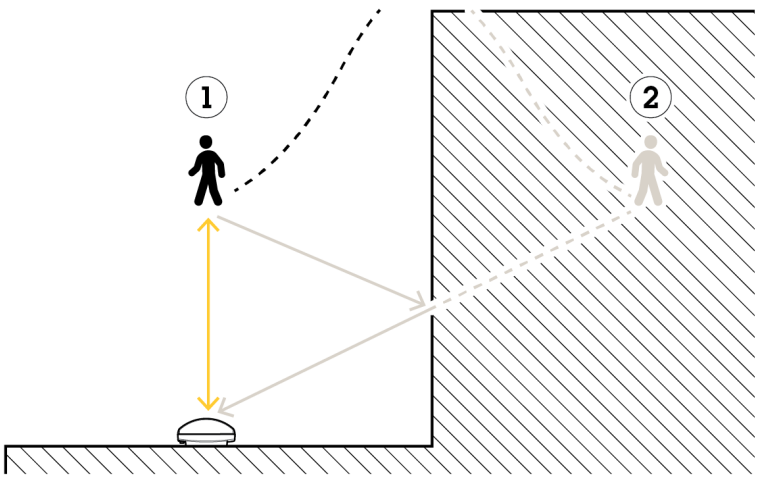


Figure 4.4: The radar signal takes two paths back to the radar after reflecting on the target which makes the system track a person inside the wall. Known as the multipath phenomenon. This could also happen near other reflective things such as the floor or a table.

To reduce detections from the multipath problem we create a map of every room that we record in and then filter out all points which are outside the walls. The map is created by walking around the perimeters of the room, recording two (x, y) points for each wall and defining the wall as a line in the xy-plane. We then filter out all points which are in a wall.

One thing to note is that this method does not necessarily fix the multipath problem in the case that the radar signal travels from the wall to the person and back, but we did not find this to be a significant problem.

4.2.3 Frame Squashing

As the radar which was used detects moving targets, there are times when people do not move enough to produce a sufficient amount of detections. Examples of this can be when a person is sitting still or lying down. If the number of detections is too low no classification can be performed since the classifiers require a certain number of points to function. To resolve this we perform so called frame squashing, which means that several raw frames are squashed into one frame before it is clustered and classified. The squashing is done adaptively. This means that if a frame contains lots of points, few frames are squashed and if a frame contains few points, more frames are squashed together.

We set the threshold at 300 points, so if a frame has fewer than 300 points more frames will be squashed together. This frame squashing will continue until one "squashed" frame contains more than 300 points or if more than 10 frames have been combined without

reaching the threshold. In the case that the threshold was not reached within 10 frames it is deemed that no classification can be done at that point in time. One potential problem with this is that the classification could become harder as a person walking across the scene would produce a "smudged out" point cloud. When testing we see that frame squashing mostly happens if a person does not move in the scene, in which case "smudging" would not be a problem.

We split this adaptive frame squashing strategy into two different flavors: frame binning and frame windowing. The difference is that frame binning places the raw frames into separate bins while frame windowing works like a windowing function, producing output with some overlapping points, see Figure 4.5.

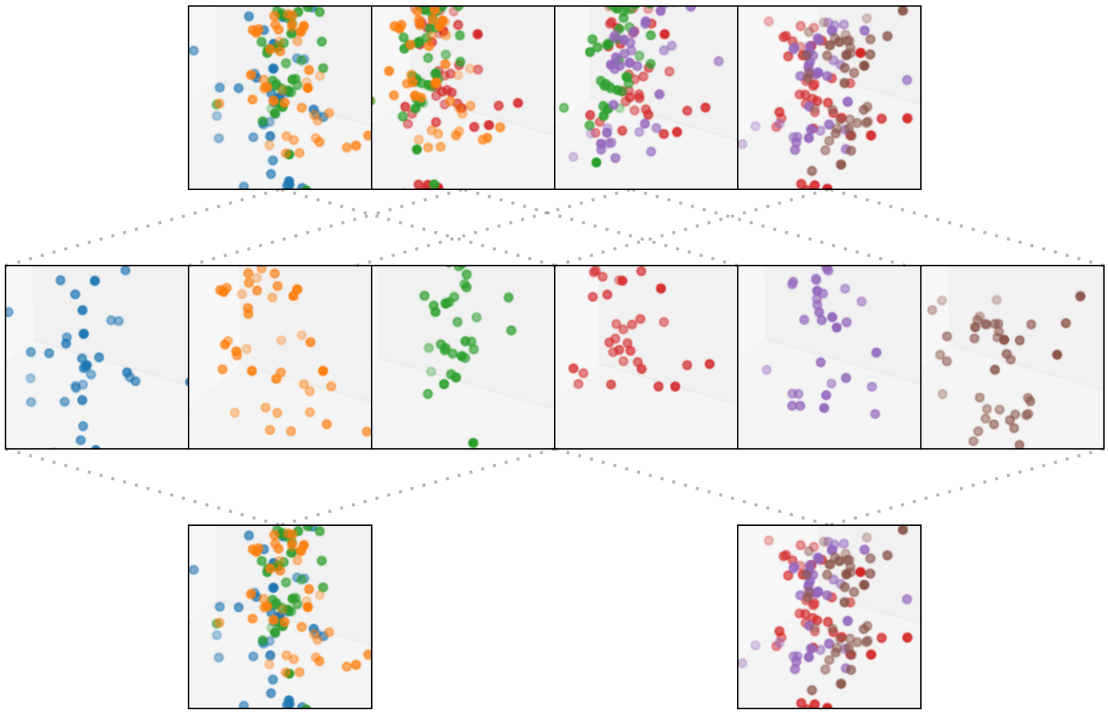


Figure 4.5: The two frame squashing techniques, binning and windowing, visualized. The middle row represents the ordinary point cloud, windowing is visualized in the upper part and binning in the lower part. In this example, three frames are squashed together each time, but in practice this number will depend on the number of points that the squashing system gets as input.

4.3 Clustering

The next part in the processing pipeline is clustering. This is done with DBSCAN. For the implementation of the DBSCAN we used the DBSCAN-module from the popular open-source Python module Scikit-learn [28] which only needs the parameters `MinPoint` and `Eps`. This implementation tries to find the distance between points using a tree-based search algorithm when possible to speed up calculations.

For the choice of the parameters, we set `Eps` to be 0.55 m and `MinPoint` to 15. The choice of these parameters was informed by looking at different point clouds and the resulting cluster.

Besides choosing the parameters, it is also possible to select which distance function is used as a metric when running the algorithm. A few different options were tested but the euclidean norm yielded the best results.

4.4 Tracking

Tracking of clusters is done to follow the positions of people in a scene over time. Since the data used for both training and evaluation was only recorded with one person in the room tracking was not needed to evaluate the models. A simple tracker was however implemented to show that the classification pipeline is able to track multiple people in the scene. The implementation of the tracker can be found in Section A.2 in the Appendix.

4.5 Classification

The classification models were trained on a set of data containing 11476 samples. This set was then divided into a training set consisting of 75% of the samples and a validation set with 25% of the samples. Each sample consisted of a point cloud with at least 128 points. The samples were obtained by using the frame windowing method described in Section 4.2.3, meaning that there was some overlap between samples. To avoid overlap between the training and validation sets the data is first split into chunks in such a way that there was no overlap in the data between chunks. These chunks are then placed into the validation and training sets, creating two sets with no overlap. For more details on the data sets, see Section 3.2.1.

4.5.1 Model 1 - Naive Classifier

The Naive classifier is meant to be a straightforward approach which the machine learning models are compared against. It looks at the general direction of the point cloud in space and determines how vertical the shape is. We assume that the point cloud is vertical when the subject is standing and more horizontal when he/she is lying down. If it is somewhere in between, the person is deemed to be sitting.

To find how vertical the cloud is, the largest principal component of the point cloud is found by performing singular value decomposition. The magnitude of the z-coordinate, $\hat{z} \in [0, 1]$, of the largest principal component determines how vertical the cloud is. This is illustrated in Figure 4.6.

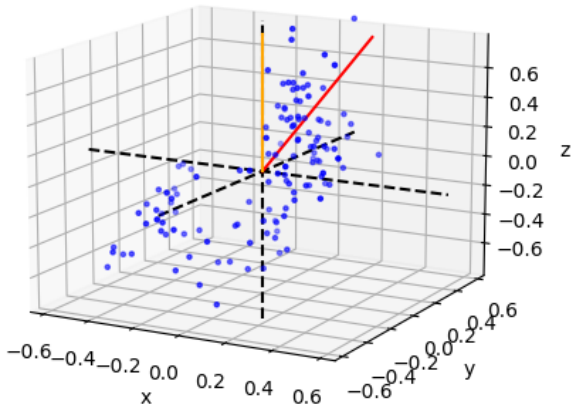


Figure 4.6: An example of the PC1-vector (red) for a point cloud that shows the direction of most variance in 3D-space. The magnitude of the PC1 in the z-direction (orange) determines how vertical the cloud is. This is the only feature used in the Naive Classifier and it is also used as one of the features in the Random Forest Classifier.

The limits for the different classes was determined by looking at the box plot of the training data in Figure 5.3 and was determined as,

- $0.9 < \hat{z} \rightarrow$ Standing;
- $0.41 < \hat{z} \leq 0.9 \rightarrow$ Sitting;
- $\hat{z} \leq 0.41 \rightarrow$ lying down.

4.5.2 Model 2 - Random Forest Classifier

When implementing the Random Forest Classifier we used the pre-built model from Scikit-learn [28]. This model has the necessary functions to train and use a Random Forest Classifier to do predictions. Outside of this it has many parameters that can be selected to affect the training of the trees.

Feature selection

Since the Random Forest Classifier makes classifications based on a feature vector, the selection of features is of utmost importance. Firstly, a number of features which we thought could be used to classify the point cloud were implemented. Then the features were dropped one at a time and if the accuracy on the validation set did not decrease that particular feature was removed.

This feature selection process resulted in the following four features:

- magnitude of z-value in PC1 (The same feature as the Naive Classifier).
- z-spread: the difference between the largest and smallest z-value.
- r-spread: the largest distance that a point has from the center of mass in the xy-plane.
- $\dot{r}$: the largest radial velocity in the point cloud.

4.5.3 PointNet Classifier

PointNet was implemented using the open source package TensorFlow [29]. A number of different configurations were tested with and without the T-net module. The number of parameters in the layers was scaled down until the validation accuracy dropped to make the network smaller while still retaining its performance.

Feature selection

Since PointNet is designed to work on point clouds with an arbitrary number of point-wise features we need to choose which features to include. The features that were found to help the model was the location in space (x, y, z) and the radial velocity v_r. Other features like signal-to-noise ratio did not improve the accuracy on the validation set.

Another parameter which must be chosen is the input size of PointNet, i.e. the number of points which is fed into the network. We would like this number to be as high as possible to give the network more information while still being able to classify most clusters (which could be too small for classification). Input size was set to 128 as it was deemed to balance these two factors. Another thing to note is that increasing the number of points also increases the computational complexity.

Pre-processing

With some pre-processing it is possible to increase the performance of the PointNet classifier. Before classifying a point cloud it is centralized around the origin. This is done to remove any spatial dependence: a point cloud should have the same classification no matter where in the room it is located. Additionally, all features of the point cloud are normalized to $[-1, 1]$. This is done to both ensure that all features have the same scale but also to make the model invariant to changes in height: we would like the model to work on people of all heights. One thing to note is that the x, y and z features are normalized with the same normalization constant, taken as the maximum magnitude of all x, y and z coordinates in the point cloud. This makes the point cloud retain its shape even after normalization.

Down-sampling is needed if the point cloud contains more points than the input-size of the network. To ensure that the down-sampled point cloud is representative of the original we utilize the internal order that the data from the radar has. It is sorted in increasing range meaning that choosing points evenly from this point-list creates a down-sampled point cloud which will be evenly sampled in range, although randomly sampled in azimuth and

elevation. This differs from the original paper where the authors use the Farthest Point Sampling-algorithm [8], which is computationally expensive.

Training

The training was done with the Adam optimizer [23] with parameters $\alpha = 0.00006$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$. During training the model is evaluated on the validation data and saved if it performs better then the current best model. Batch normalization [30] is used for every layer.

4.5.4 Model 3 - PointNet with T-Net

This model uses PointNet with the same number of layers as the model from the original paper. The sizes of the layers are scaled down by different factors to reduce model complexity as we have fewer classes and less data. The shared MLP-layers are scaled down by a factor of 2 while the other parts are scaled down by a factor of 8. This is done to reduce the number of parameters in the network while still retaining a large global feature vector which was shown to increase classification performance in the original paper [8].

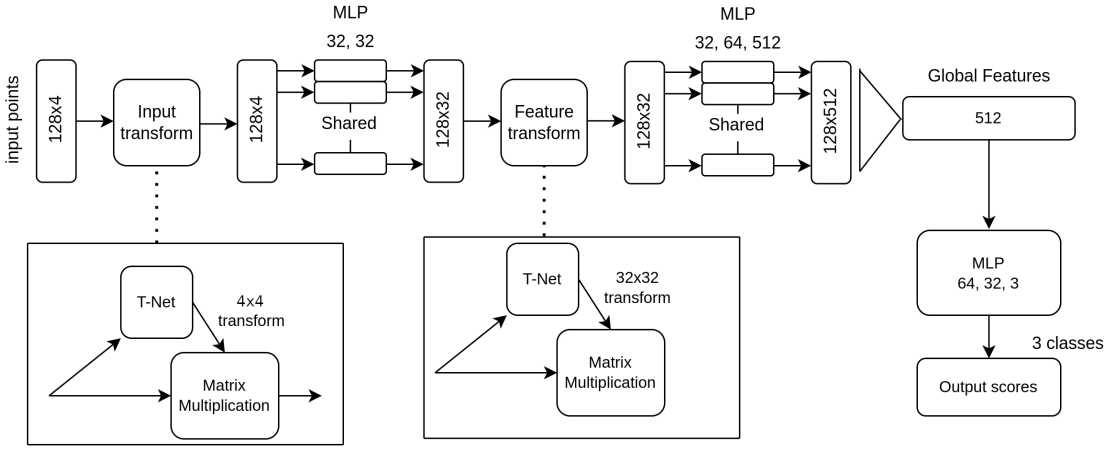


Figure 4.7: The PointNet architecture used for model 3 which is a scaled down version of the original PointNet. The Shared MLP layers are scaled down by a factor of 2 and the final MLP is scaled down by a factor of 8. The total number of parameters in this model is 137,667.

4.5.5 Model 4 - PointNet without T-Net

This model is a modified version of the PointNet architecture which does not include the T-Net parts. In the original paper T-Nets were included to make the classification of a point cloud independent of its orientation in space [8]. In the setting of our model, the orientation of a point cloud matters; rotating a point cloud of a person who is standing can create a point cloud which is lying down.

In addition to removing the T-Nets, the number of layers in the Shared MLPs is also

reduced. This was done to reduce the size of network as much as possible without losing accuracy.

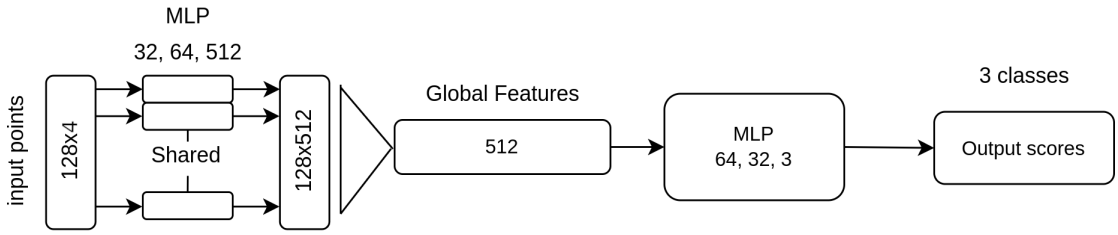


Figure 4.8: The architecture used for model 4 that uses part of the PointNet structure. The T-Net parts are removed and the Shared MLP is reduced to 3 layers. The layers are scaled down in the same way as in model 3. The total number of parameters in this model is 73,379.

4.6 Temporal models

Since the radar outputs 10 frames per second there are some pose patterns that can immediately be discarded. One example is the sequence (standing, lying down, standing) which would correspond to the subject being upright, lying down for 100 ms to then be standing again (which is not physically possible). In an effort to capture these types of temporal effects we introduce models which operate on sequences of classifications.

4.6.1 Rolling Average

The most basic temporal model just sums up the k latest prediction probabilities and takes the largest of these as its prediction. This should make the classification more stable with smoother transitions that are less affected by noise. The parameter k was chosen to be 5 as this represents the last 0.5s of data. This is enough to smooth out flickering while not delaying the actual transitions too much.

4.6.2 Hidden Markov Model

A Hidden Markov Model is implemented with three states and three possible sensor readings corresponding to the three poses: standing, sitting and lying down. The sensor model, which consists of probabilities,

$$P(e_t = j | X_t = i) = \frac{P(e_t = j, X_t = i)}{P(X_t = i)},$$

for each combination of sensor reading e_t and actual state X_i will then have 9 elements which must be determined. To find these we utilize a confusion matrix C obtained from predicting on the validation set. Each element C_{ij} contains the number of samples which true pose was i and the classification model predicted as j. By normalizing C over each row the elements C_{ij} will then correspond to the sought after probability $P(e_t = j | X_t = i)$.

Since the optimal values in the transition matrix are highly dependent on the setting in which the model is deployed in we simply set these with the following assumptions:

- A person changes pose every 20 seconds.
- The probability of switching from pose i to j is the same as switching from pose j to i.
- Switching from sitting to standing is 4 times less likely compared to going from standing or lying down to sitting.

These assumptions gives us the following transition matrix,

$$T = \begin{pmatrix} 0.9995 & 0.0004 & 0.0001 \\ 0.00025 & 0.9995 & 0.00025 \\ 0.0001 & 0.0004 & 0.9995 \end{pmatrix}. \tag{4.1}$$

4.7 Evaluation

This section describes what experiments were done in order to answer the questions in Section 1.2.

The first question is:

"How should point cloud data from a high resolution radar be processed to enable classification of human poses?"

This is quite difficult to answer. Since the pipeline clusters point clouds before feeding them into a classifier one way to evaluate the data processing part would be to evaluate the clusters that it produces. The ideal way of doing this would be to annotate each point in the training point clouds as either noise or originating from a human and then quantify the quality of the clusters. Doing this sort of annotation process on a three-dimensional point cloud is very labor intensive which is why it is not done in this thesis.

Instead, the data processing chain is evaluated qualitatively by visualizing the feature space to see how separated the different classes are. In addition to this one can argue that the classification results also reflect the data processing quality. If the data processing works poorly the classification results will not be good. This is especially true for the naive classifier since it cannot learn and compensate for any faults in the data processing.

The two other questions,

"Can a model using a Random Forest Classifier reliably classify the pose of a human?"

and

"Can a model using PointNet reliably classify the pose of a human?"

are easier to evaluate by comparing the predicted poses with the actual poses. This is done for all samples in the test set which was recorded in a different location and with different people. To give a simple scalar score which describes how the models perform a number of different metrics are used.

4.7.1 Evaluation metrics

To measure the performance of the classification models precision, recall and F1-score are used. These metrics can be applied for each class to see how the each model performs on certain classes but can also be averaged over all classes to get an evaluation for how well the model performs in total on the task. In addition accuracy is calculated which is just the number of predictions which are correct divided by the total number of samples.

Precision measures how many predictions of a class are actually of that class and is given as,

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}. \tag{4.2}$$

Recall measures how many samples of a given class the model is able to classify correctly. The formula for recall is,

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}. \tag{4.3}$$

The F1-score combines precision and recall into one score which is nice for overall comparison. This metric is given as the harmonic mean of precision and recall and is calculated as ,

$$\text{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{4.4}$$

Some tasks require higher scores for recall or precision specifically, but in this thesis the more general metrics F1-score and accuracy are used to compare models.

Chapter 5

Results

5.1 Classification per frame

This section contains the results from the frame-wise classification results. Each model is evaluated on the test set described in Section 4.5.

5.1.1 Model 1 - Naive classifier

The classification results for model 1 can be seen in Table 5.1. Since this naive classifier only looks at how vertical the shape is, there is bound to be some errors due to overlap between the classes. This overlap is visualized in Figure 5.3.

Class	Precision	Recall	F1	Support
Standing	0.872	0.899	0.885	1626
Sitting	0.684	0.512	0.586	1124
Lying down	0.650	0.851	0.737	750
Weighted average	0.764	0.765	0.757	3500
Accuracy	76.5%			3500

Table 5.1: The results of classification on the test set using the naive classifier from Section 4.5.1. It performs better on the standing and lying down classes as they are centered more towards each end of the feature interval. Overall this is a rather good score for such an easy model.

5.1.2 Model 2 - Random Forest Classifier

To decide on what the best parameters would be for training the Random Forest Classifier, a grid search was done for the parameters `n_estimators`, `max_depth`, `max_leaf_nodes` and `max_samples`. For each combination of parameters, the model was trained on the training set described in Section 4.5 and then tested on the validation set. The optimal values found by this grid search are found in Table 5.2 below. All the parameter values that were used in training can be found in Table A.1 in the Appendix.

Parameter name	Value
n_estimators	150
max_depth	7
max_leaf_nodes	60
max_samples	0.8

Table 5.2: The parameter values obtained from grid search on the Random Forest Classifier.

The frame-wise classification results for the Random Forest Classifier are shown in Table 5.3. Compared to the naive classifier, this model performs better overall. The biggest increase is found in the sitting class where the Random Forest Classifier has a considerably better F1-score.

Class	Precision	Recall	F1	Support
Standing	0.949	0.935	0.942	1626
Sitting	0.795	0.668	0.726	1124
Lying down	0.683	0.867	0.764	750
Weighted average	0.842	0.835	0.834	3500
Accuracy	83.5%			3500

Table 5.3: The results of classification on the test set using the Random Forest Classifier. The scores improved significantly compared to the naive classifier.

5.1.3 Model 3 - PointNet

The model was trained with the following parameters:

- Optimizer: Adam;
- Learning rate (α): 0.0006;
- β_1: 0.9;
- β_2: 0.999;
- ϵ: 10^{-7};
- batch size: 64;
- epochs: 60.

To train a more generalized model, a jittered version of each point cloud was added to the training set. The learning curve with the loss and categorical accuracy for both the training and validation set can be seen in Figure 5.1.

The results of the frame-wise classification on the test set can be found in Table 5.4. The scores are similar to those on model 2. Both models perform better on the standing class and a little worse on the sitting and lying down classes. The added complexity of this

model did not increase the performance compared to the less complex Random Forest Classifier.

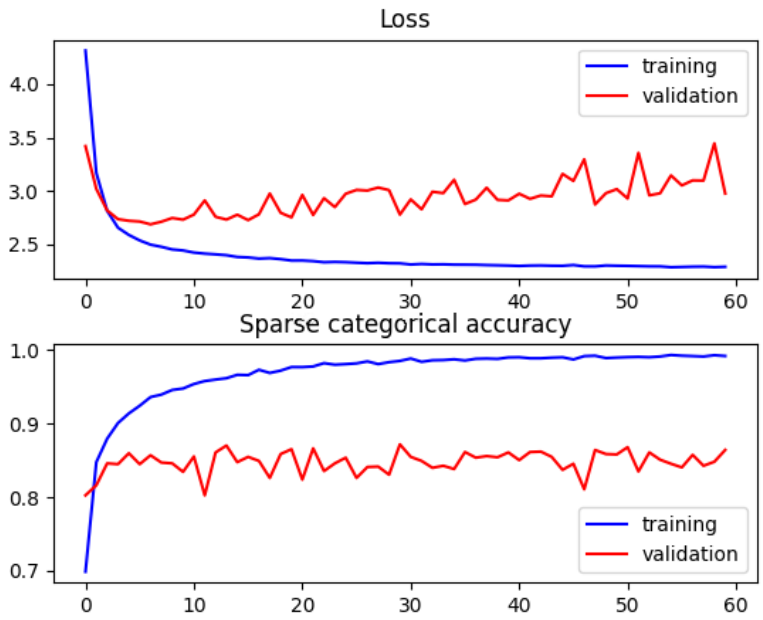


Figure 5.1: The loss and categorical accuracy when training model 3. The validation loss indicates that the model is in its most general form after around 5-7 epochs.

Class	Precision	Recall	F1	Support
Standing	0.915	0.938	0.927	1626
Sitting	0.759	0.716	0.737	1124
Lying down	0.757	0.779	0.768	750
Weighted average	0.831	0.833	0.832	3500
Accuracy	83.3%			3500

Table 5.4: The results of classification on the test set with model 3 that is a scaled down version of the original PointNet-structure described in Section 4.5.4. This model performs about as well as the simpler Random Forest Classifier.

5.1.4 Model 4 - PointNet without T-Net

The model was trained with the following parameters:

- Optimizer: Adam;
- Learning rate (α): 0.0006;
- β_1: 0.9;
- β_2: 0.999;

- ϵ: 10^{-7};
- batch size: 64;
- epochs: 60.

To train a more generalized model a jittered version of each point cloud was added to the training set. The learning curve with the loss and categorical accuracy for both the training and validation set can be seen in Figure 5.2.

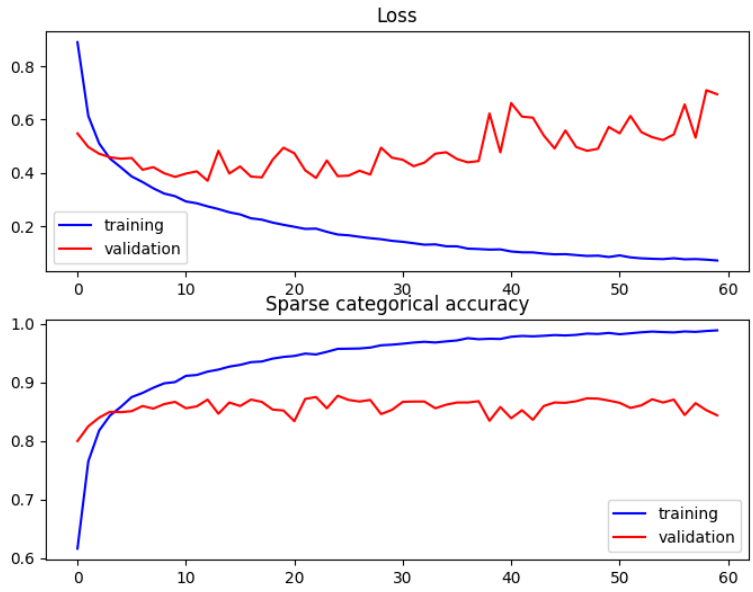


Figure 5.2: The loss and categorical accuracy when training model 4. The validation loss is minimized after around 10-15 epochs. After this the model tends to be overfitted to the training data and should not perform as well on other datasets.

The classification results on the test set for model 4 can be seen in Table 5.5. There is an improvement in the F1-score for the sitting class and the lying down classes compared to the Random Forest-model. The F1-score for standing are about the same. Removing the T-Net part was effective for this problem. This could either be because there is information in the rotation that is lost in the T-Net or because the more complex model 3 does not generalize as well.

Class	Precision	Recall	F1	Support
Standing	0.927	0.959	0.943	1626
Sitting	0.809	0.748	0.777	1124
Lying down	0.789	0.817	0.803	750
Weighted average	0.859	0.861	0.860	3500
Accuracy	86.1%			3500

Table 5.5: The results of classification on the test set with model 4. This model used the scaled down version of PointNet without the T-Net parts and fewer layers in the shared MLP, as described in Section 4.5.5. This model performs the best out of the proposed models. It maintains a high score for the standing class while improving on the sitting and lying down classes.

5.2 Temporal models

The two different temporal models were tested on the classifier with the best accuracy, Model 4. The test was done using the same test set, but classified in order so that the temporal models could use the output from the frame-wise classifier.

5.2.1 Rolling Average

With the rolling average on Model 4, the following results in Table 5.6 were obtained:

Class	Precision	Recall	F1	Support
Standing	0.931	0.983	0.956	1626
Sitting	0.862	0.780	0.819	1124
Lying down	0.828	0.845	0.837	750
Weighted average	0.887	0.888	0.887	3500
Accuracy	88.8%			3500

Table 5.6: The results of classification on the test set with a Rolling Average applied on predictions from Model 4. The window size on which the average was taken was set to five frames.

5.2.2 Hidden Markov Models

In terms of accuracy, Hidden Markov Models gave a slightly worse result compared to rolling average, see the following Table 5.7

Class	Precision	Recall	F1	Support
Standing	0.935	0.976	0.955	1626
Sitting	0.839	0.789	0.813	1124
Lying down	0.828	0.822	0.825	750
Weighted average	0.881	0.883	0.882	3500
Accuracy	88.3%			3500

Table 5.7: The results of classification on the test set with a Hidden Markov Model applied on predictions from Model 4.

Both temporal models improve the accuracy compared to the frame-wise classification of Model 4 (see Table 5.5). Rolling average gives an improvement of 2.7 percentage points while Hidden Markov Models raises the accuracy with 2.2 percentage points. One thing to note is that this increase in accuracy comes largely from increases better performance on the sitting and lying down classes.

5.3 Visualizing the feature spaces

5.3.1 Model 1 - Naive classifier

For the naive classifier the single feature for a point cloud was how vertical the point cloud was. As this is a 1-dimensional space it can be represented by a box plot for the different classes in the data, as can be seen in Figure 5.3.

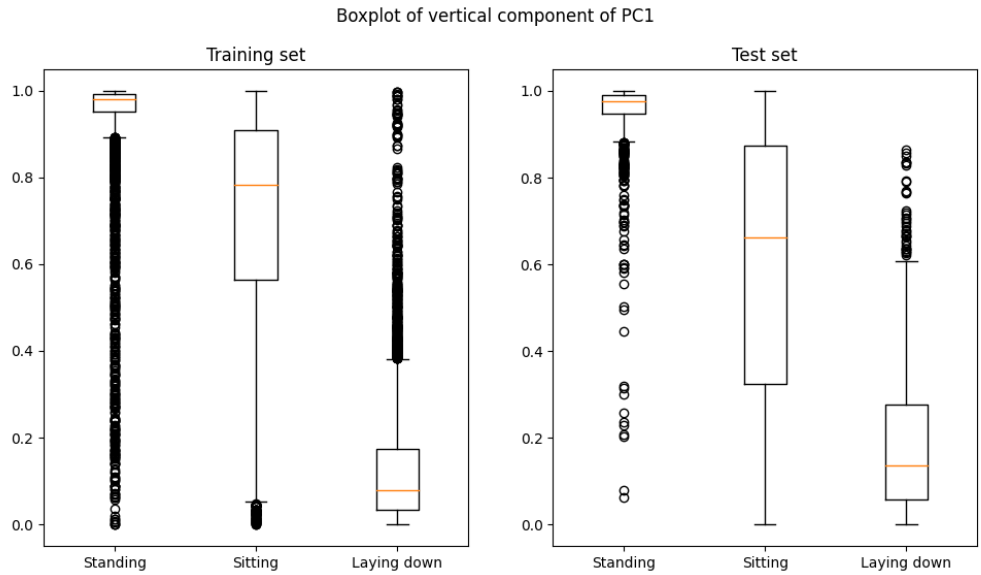


Figure 5.3: The box plot of vertical score for the samples of different classes in the training set and test set, as described in Section 4.5.1. This is the variable used in the naive classifier. There are overlap in the quartiles between the sitting class and the two other classes leading to the miss-classifications.

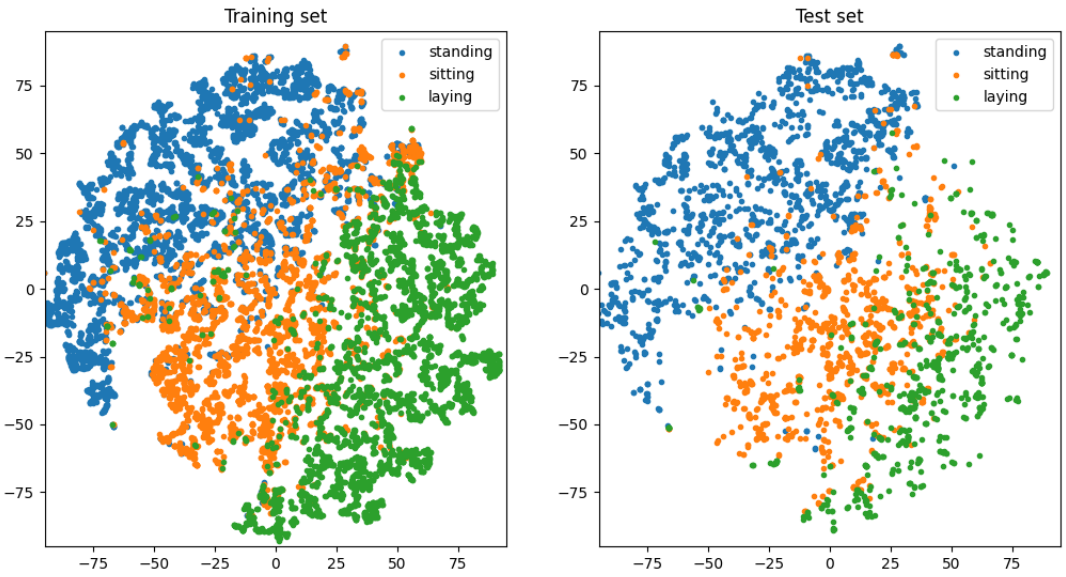


Figure 5.4: The t-SNE visualization of the 4D feature space used as input to the Random Forest Classifier. Blue represents standing samples, orange represents sitting samples and green represents samples of lying down. All the data was fitted together and then plotted separately. The left plot shows the training set and the right shows the test set. Overall the different classes get separated but there are some samples from sitting and lying down that are scattered all around the plot.

5.3.2 Model 2 - Random Forest Classifier

For model 2 the t-SNE transformation was used to transform the 4D feature space into a 2D feature space. The results of this can be seen in Figure 5.4. Using PCA would probably be a good option here instead, but t-SNE was used for a fair comparison to the other models. The feature space seems rather separated but there are some overlap in the edges. The test set seems to differ more in the edge between sitting and lying down compared to the training set.

5.3.3 Model 3 - PointNet

For model 3 the t-SNE transformation was used to map the 512 global feature vector into a 2D-space. The result of this transform can be seen in Figure 5.6. The model seem to separate the data in the training set well, but a bigger part of the test set seem to be mixed with each other. This points towards the model being overfitted to the training data, limiting its performance on the test set.

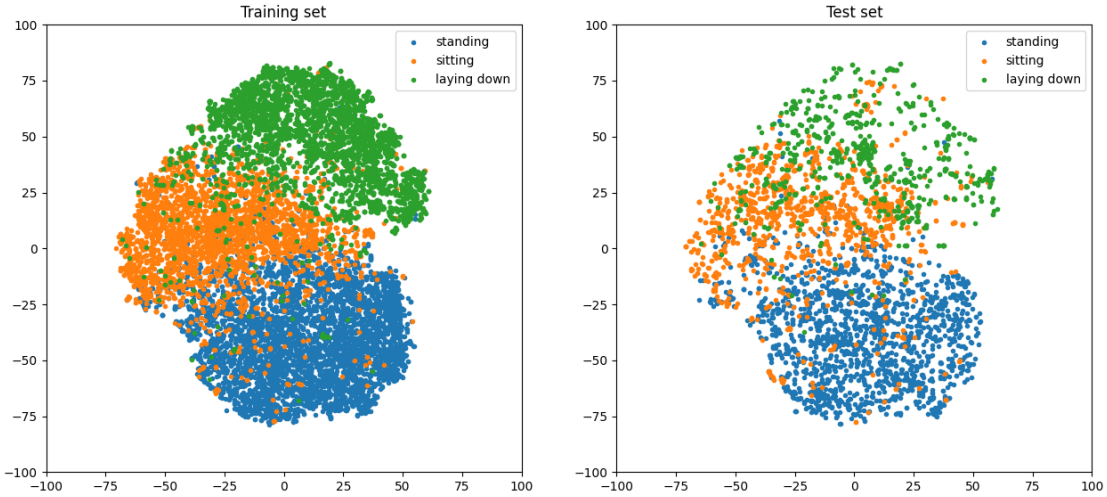


Figure 5.5: The t-SNE visualization of the global feature vector from model 3. Blue represents standing samples, orange represents sitting samples and green represents samples of lying down. All the data was fitted together and then plotted separately. The left plot shows the training set and the right shows the test set. The training set seems to be separated quite well, while the test set looks to be more mixed. This could be caused by an overfit to the training data or it could be due to actual differences in characteristics of the data.

5.3.4 Model 4 - PointNet without T-Net

For model 4 the t-SNE transformation was used to map the 512 global feature vector into a 2D-space. The result of this transform can be seen in Figure 5.6. It does not seem to separate the classes that well, as the shape of the clusters seem more flat and close to each other. But the overall shapes seem to match better between the training data and the test data compared to the other models. This is probably why this model performs better.

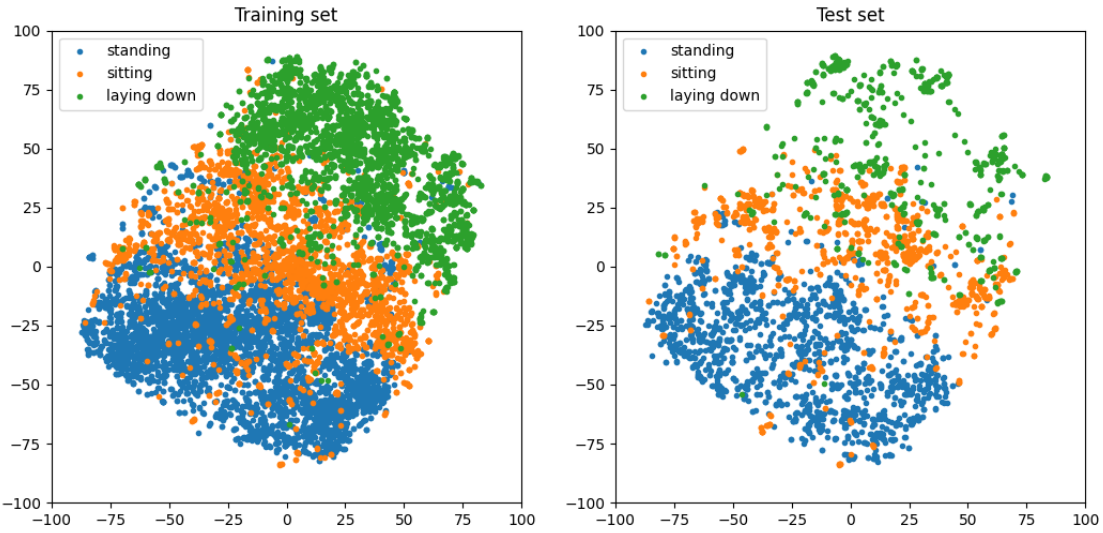


Figure 5.6: The t-SNE visualization of the global feature vector from model 4. Blue represents standing samples, orange represents sitting samples and green represents samples of lying down. All the data was fitted together and then plotted separately. The left plot shows the training set and the right shows the test set. The overall shape shows the three classes relatively separated but there are many samples that are mixed along the borders, which is unwanted.

Chapter 6

Discussion

6.1 Limitations

6.1.1 Data

The dataset used in this thesis was relatively small, with only 14976 samples, compared to the usual data sizes for deep learning. As a comparison, the ModelNet40 dataset used to train the original PointNet has over 137,000 varied samples [7]. The training history for the PointNet-based models (see Figures 5.1 and 5.2) shows that the validation loss starts low and then quickly reaches it's minimum. The validation loss is also quite noisy. We believe this is caused by the validation dataset being small and not varied enough. Since we choose the model with the lowest loss on the validation data it is possible that the selected model is underfitted if the validation data is not representative of the general problem, which seems to be the case. One solution to this problem would be to obtain more data.

All the models have trouble with the sitting and lying down classes. This could be due to the fact that these poses were more prone to occlusion of body parts and splitting of the cluster caused by the subject not moving enough. One solution to this problem would be to detect that the target is not moving enough to produce a good classification and just keep their pose as is. This should work since if a person is not moving, he or she will not be able to switch poses.

Comparing the training data with the test data in the plotted feature spaces in Section 5.3, some differences between the two sets can be found. One thing that can be seen is that the test set has more samples from the sitting class that overlap more with the lying down class. This phenomenon could be explained by the fact that the test and training set are recorded on different people that have slightly different sitting and lying down positions. We believe that the models could handle this problem with more varied data.

6.1.2 Model Robustness

The few articles that we have found concerning human activity recognition on radar point clouds [12, 31] record their subjects on a fixed distance in front of the radar. While our

model has both fewer classes and lower accuracy our work shows that human behavior can be tracked by a radar in realistic scenarios where occlusions, reflections and noise are large problems. Additionally it must be stressed that the test set that was used to evaluate our model was recorded on subjects which the model had not seen before in a new environment with a different furniture. This suggests that the classification could generalize to even more kinds of spaces and people. To be sure of this, more tests would have to be done.

6.1.3 Dependency on clustering

The later part of the processing-pipeline depends on the filtering and clustering part of the data processing. If the point cloud of a target is not fully captured or contains too much noise, its characteristics could differ very much from a point-wise ground truth. There is no quantitative way to measure this with the current annotations. The only way would be to look frame-by-frame at clustered and non-clustered point to compare what seems to be right.

6.2 Computational complexity

For real-life applications the computational cost should be considered for the different models along with the classification results for deciding what is a better choice for the application. With our implementation in Python, a quick test showed that both of the PointNet-based models was able to make predictions in about half the time it took for the Random Forest classifier. Between the two PointNet-models, the smaller version decreased the prediction time by about 20%.

The PointNet model scales with the different number of parameters in the layers as well as the number of input points. The Shared MLPs before the max-pooling have to be inferred on every point which causes a lot of computation. Making these layers smaller has a bigger effect on the computational cost compared to the MLP after the global feature vector. The number of input points could be decreased to save computation, but this would also mean less information in total. The original paper on PointNet shows that having more input points generally give better results [8].

6.3 Hidden Markov Model Limitation

Both the temporal models improved the performance of the frame-wise classification model. A surprising result was that the very simple rolling average technique beat the more complex Hidden Markov Model. There are a number of things which could explain this.

Firstly, the transition matrix is just hard-coded based on some assumptions. Instead, it could be adapted continuously adapted during predictions with a expectation-maximization algorithm [25]. Secondly, it can be argued that the Markov Assumption,

$$P(X_{n+1} = j \mid X_n = i_n, ..., X_0 = i_0) = P(X_{n+1} = j \mid X_n = i_n), \tag{6.1}$$

does not hold true for tracking the pose of a person over time. This can be shown by looking at the transition probabilities,

$$P(X_{n+1} = \text{standing} \mid X_n = \text{lying down}, X_{n-1} = \text{standing}) \tag{6.2}$$

and,

$$P(X_{n+1} = \text{standing} \mid X_n = \text{lying down}, X_{n-1} = \text{lying down}). \tag{6.3}$$

According to the Markov Assumption (6.1) these two probabilities should be the same. By looking at the pose sequences this is obviously not true, though. The pose sequence (standing, lying down, standing) implies that the person stands, lays down for 100 ms and then stands again. This is not physically possible so (6.2) should zero. On the other hand the pose sequence described in (6.3) corresponds to a person lying down and then standing. This is definitely possible so (6.3) should be non-zero. Therefore the the Markov Assumption does not hold for describing poses over time, given that the poses are classified 10 times per second.

This is a problem since it shows that the original assumption on which the HMM builds upon does not hold. There are some quite straight-forward solutions to this though. One alternative is to expand the state-space of the Markov Chain to include pose pairs (or pose triplets) instead of single poses. This way the transition matrix could be adapted to exclude sequences which are not physically possible.

6.4 Future Work

The classification models in this thesis were built upon previous work in point cloud classification with some modifications for radar data. There was not enough time to test all the different ideas we had, as well as other models from previous works. Given more time, we think the model could be considerably improved upon.

6.4.1 Data

As mentioned before, a bigger data set would help the deep learning models find a more generalized model. The easiest way to do this would be to just record and annotate more data. Another way to generate more data would be to improve the data augmentation. Rotating the point cloud could help with the fact that people sit and lay down in different directions. However, this augmentation would have to be implemented carefully due to how the radar works. Rotating the cloud too much could result in unrealistic point clouds compared to an actual recording at that angle. Another type of augmentation that could help would be some type of mirroring in space.

6.4.2 Segmentation and clustering

Selecting what is a target and what is noise is important for the following part of the model. In this work there was no ground truth which could be used to evaluate the performance of this step. The effects of propagating errors could therefore not be estimated.

With a point-wise ground truth there are a couple of options that could be improved. The first idea would be to use another PointNet-based model to do semantic segmentation. This

has been done in the original paper [8] and its follow-up PointNet++ [32] which showed promising results. Of these networks, PointNet++ would probably be the better choice as it captures both local and global features. For point clouds of the entire rooms, the local structures would hold much of the information for the segmentation. This segmentation could either just find target and non-target points to filter more points before clustering or it could do pointwise classification to merge the different steps.

6.4.3 Classifiers

Other types of classes could be added to find whether the given cluster is human or not. The model assumes that every cluster is a person, which is not always the case in the indoor environments. Examples of classes that could be added here are pet animals or a class which represents everything else. Furthermore, it would be good to see if the pose classification could be extended to include activities, such as walking, running, etc. This could be done as a new task or by combining the information of the pose together with the speed of the track and amount of movement within the cluster.

A PointNet-based approach was used instead of PointNet++ since the point cloud from the radar is not as accurate as data from CAD-models or some lidars. But the increased information from local features in PointNet++ could improve the classification. To determine if that is the case, a model using PointNet++ would need to be compared to the models presented in this thesis.

Chapter 7

Conclusion

The proposed classification pipeline was able to both find people in the point cloud data and classify their poses into three categories: standing, sitting or lying down. The model was evaluated in an environment and on people which the model had not seen before, suggesting that the model could generalize into other indoor settings.

The proposed model is able to reliably classify a person as standing but scores a bit lower for the sitting and lying down classes. This can be attributed to the model confusing sitting with lying down and vice versa. We believe an underlying reason for this shortcoming is the data processing where clusters from people sitting or lying down were more prone to be incomplete. This tends to happen when the person is already sitting/lying down and not moving as much. As long as the moment when the person assumes the position is classified correctly, this is a problem that can be overcome.

Given a bigger and more varied dataset together with the proposed improvements, we believe the classification models using PointNet could become very reliable in keeping track of a person and its poses over time. With an improved model, this type of set-up of a radar could prove to be a good alternative in monitoring a person in areas where a camera should not be used.

Bibliography

[1] Fahad Jibrin Abdu, Yixiong Zhang, Maozhong Fu, Yuhan Li, and Zhenmiao Deng. Application of deep learning on millimeter-wave radar signals: A review. *Sensors*, 21 (6), 2021. ISSN 1424-8220. doi: 10.3390/s21061951. URL https://www.mdpi.com/1424-8220/21/6/1951.

[2] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual MLP framework. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=3Pbra-_u76D.

[3] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4338–4364, 2021. doi: 10.1109/TPAMI.2020.3005434.

[4] Weiping Liu, Jia Sun, Wanyi Li, Ting Hu, and Peng Wang. Deep learning on point clouds and its application: A survey. *Sensors*, 19:4188, 09 2019. doi: 10.3390/s19194188.

[5] Saifullahi Aminu Bello, Shangshu Yu, Cheng Wang, Jibril Muhmmad Adam, and Jonathan Li. Review: Deep learning on 3d point clouds. *Remote Sensing*, 12(11), 2020. ISSN 2072-4292. doi: 10.3390/rs12111729. URL https://www.mdpi.com/2072-4292/12/11/1729.

[6] David Griffiths and Jan Boehm. A review on deep learning techniques for 3d sensed data classification. *Remote Sensing*, 11:1499, 06 2019. doi: 10.3390/rs11121499.

[7] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d ShapeNets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. doi: 10.1109/cvpr.2015.7298801. URL https://doi.org/10.1109/cvpr.2015.7298801.

[8] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2016. URL https://arxiv.org/abs/1612.00593.

[9] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and

Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[10] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *International Conference on Computer Vision (ICCV)*, 2019.

[11] Jie Bai, Kai Long, Sen Li, Libo Huang, and Lianfei Dong. Multi-objective classification of three-dimensional imaging radar point clouds: Support vector machine and pointnet. *SAE International Journal of Connected and Automated Vehicles*, 4 (12-04-04-0028), 2021.

[12] Akash Deep Singh, Sandeep Singh Sandha, Luis Garcia, and Mani Srivastava. Radhar: Human activity recognition from point clouds generated through a millimeter-wave radar. In *Proceedings of the 3rd ACM Workshop on Millimeter-Wave Networks and Sensing Systems*, mmNets'19, page 51–56, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369329. doi: 10.1145/3349624.3356768. URL https://doi.org/10.1145/3349624.3356768.

[13] Anish Shastri, Neharika Valecha, Enver Bashirov, Harsh Tataria, Michael Lentmaier, Fredrik Tufvesson, Michele Rossi, and Paolo Casari. A review of indoor millimeter wave device-based localization and device-free sensing technologies, 2021. URL https://arxiv.org/abs/2112.05593.

[14] Jacopo Pegoraro and Michele Rossi. Real-time people tracking and identification from sparse mm-wave radar point-clouds. *IEEE Access*, 9:78504–78520, 2021. doi: 10.1109/ACCESS.2021.3083980.

[15] James A. Scheer Mark A. Richards and William A. Holm (Editors). *Principles of Modern Radar : Basic Principles, Volume 1*. ProQuest Ebook Central, 2010.

[16] S. Suleymanov. Design and implementation of an fmcw radar signal processing module for automotive applications, August 2016. URL http://essay.utwente.nl/70986/.

[17] Sofia Catalucci and Nicola Senin. State-of-the-art in point cloud analysis. In *Advances in Optical Form and Coordinate Metrology*, 2053-2563, pages 2–1 to 2–48. IOP Publishing, 2020. ISBN 978-0-7503-2524-0. doi: 10.1088/978-0-7503-2524-0ch2. URL https://dx.doi.org/10.1088/978-0-7503-2524-0ch2.

[18] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[19] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[20] Kevin Gurney. *An Introduction to Neural Networks*. Taylor & Francis, Inc., USA, 1997. ISBN 1857286731.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[22] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, cop., New York, NY, 2006. ISBN 9780387310732.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[25] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4th edition, 2021. ISBN 978-1-292-40113-3.

[26] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL https://www.jmlr.org/papers/v9/vandermaaten08a.html.

[27] Karl Sigman. Lecture notes on stochastic modeling i, 2009.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[29] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/ioffe15.html.

[31] Youngwook Kim, Ibrahim Alnujaim, and Daegun Oh. Human activity classification based on point clouds measured by millimeter wave mimo radar with deep recurrent neural networks. *IEEE Sensors Journal*, 21(12):13522–13529, 2021. doi: 10.1109/JSEN.2021.3068388.

[32] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017. URL http://arxiv.org/abs/1706.02413.

Appendix A

A.1 Parameters for Random Forest Classifier-implementation

The parameters used for the implementation of the Random Forest Classifier was used given the implementation from *sklearn* and can be found below in Table A.1.

Parameter name	Value
n_estimators	150
criterion	”gini”
max_depth	7
min_samples_split	2
min_samples_leaf	1
max_features	”auto”
max_leaf_nodes	60
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
warm_start	False
class_weight	1 for all classes and inputs
ccp_alpha	0.0
max_samples	0.8

Table A.1: The different parameters used when training the Random Forest Classifier with the model from *Sci-kit learn*. The parameters were chosen by a grid search with creating a validation set from some samples in the training set and picking the model that performed the best on that set.

A.2 Tracking

To be able to link different clusters in time to the same target we had to implement a tracker. Since we did not have many targets in the scene at the same time the tracker could be simple. Psuedocode for the implemented tracker is found in Algorithm 2.

Algorithm 2 Tracker

```
1: function Track(Clusters, Tracks, dT, gate, tMax)
2:     sort(Tracks) based on lowest velocity
3:     for track in Tracks do
4:         AssignCluster(track, Clusters, dT, gate)
5:     end for
6:     if Clusters is not empty then
7:         for cluster in Cluster do
8:             newTrack = trackFromCluster(cluster)
9:             Tracks.append(newTrack)
10:        end for
11:    end if
12:    for track in Tracks do
13:        if track.missed > tMax then Tracks.delete(track)
14:        end if
15:    end for
16: end function
17:
18: function AssignCluster(track, Clusters, dT, gate)
19:    nearestCluster = Clusters[0]
20:    minDist = distance(track.center, Clusters[0].center)
21:    for cluster in Clusters do
22:        currDist = distance(track.center, cluster.center)
23:        if currDist < minDist then
24:            nearestCluster = cluster
25:            minDist = currDist
26:        end if
27:    end for
28:    if minDist < gate then
29:        track.velocity = (track.center - nearestCluster.center) / dT
30:        track.update(nearestCluster)
31:        Clusters.delete(nearestCluster)
32:        track.missedTime = 0
33:        track.new = False
34:    else
35:        track.missedTime = track.missedTime + dT
36:    end if
37: end function
```

Master's Theses in Mathematical Sciences 2022:E29
ISSN 1404-6342

LUTFMA-3478-2022

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/