Technical Report

Robin Mannberg, Martin Andersson, Emma Beskow, Ella Grundin,
Joel Nilsson, Gabriel Suihko and Jianxin Qu

December 23, 2021

Version 1.0

Status

Reviewed	The project group	2021-12-23
Approved		

Project Identity

Group E-mail:	tsks23group1@gmail.com
Orderer:	Danyo Danev Phone: +46 (0)13 28 13 35 E-mail: danyo.danev@liu.se
Customer:	Danyo Danev Phone: +46 (0)13-28 13 35 E-mail: danyo.danev@liu.se
Supervisor:	Ema Becirovic Phone: +46 (0)13-28 19 11 E-mail: ema.becirovic@liu.se
Supervisor:	Jianan Bai Phone: +46 (0)13-28 26 13 E-mail: jianan.bai@liu.se
Course Responsible:	Danyo Danev Phone: +46 (0)13-28 13 35 E-mail: danyo.danev@liu.se

Participants of the group

Name	**Role**	**E-mail**
Robin Mannberg	Project Manager	robma370@student.liu.se
Martin Andersson	Test Manager	maran594@student.liu.se
Emma Beskow	Document Manager	emmbe571@student.liu.se
Ella Grundin	Hardware Manager	ellgr825@student.liu.se
Joel Nilsson	Chief of Design	joeni078@student.liu.se
Gabriel Suihko	Graphics Manager	gabsu290@student.liu.se
Jianxin Qu	Software Manager	jiaqu952@student.liu.se

CONTENTS

DOCUMENT HISTORY

Version	Date	Changes made	Sign	Reviewer
1.0	2021-12-23	First version. Minor changes.	The project group	Supervisors
0.2	2021-12-20	Fixed grammar, added references, moved some things in Section 3.	The project group	Supervisors
0.1	2021-12-14	First draft.	The project group	Supervisors

1 INTRODUCTION

The research area of human activity recognition and detection in indoor environments based on analysis of channel state information (CSI) of radio frequency (RF) communication links has recently seen an upswing owing to the development of deep learning (DL) methods [1]. The development has grown following the failure of conventional machine learning (ML) approaches to solve more complex human activity recognition tasks. However, it is still of interest to investigate the performance of conventional ML models on less complex tasks. These models would require less samples for training and the result would be easier to interpret. Many detection and recognition methods have been developed and deployed commercially by the use of DL methods, for instance, sleep monitoring, vital sign monitoring, fall detection, localisation and tracking, activity monitoring and crowd counting [2], [3]. RF sensing provides a less intruding, less energy-consuming way to detect and recognise human activity that is independent of both light conditions and line-of-sight [4]. It is possible to integrate these solutions into existing WiFi-equipment. All of this contributes to the attractiveness of human activity recognition or detection by RF sensing as an area of research.

This project was to develop a system for detection of static (no moving object or person present) and dynamic indoor environments. The system is implemented in Python and uses several ADALM Pluto Software-Defined Radio (Pluto SDR) devices to estimate channels in an indoor environment. After collecting CSI, ML algorithms are used to classify different types of environments as either static or dynamic, where the latter category is broken down into several specific types of motion of an object or a person. The performance of the different models is then evaluated and compared.

The aim of this document is to provide the reader with a summary of previous work done within this area-of-research, provide the results of the project, in addition to conclusions drawn and design decision made for the final product.

2 PREVIOUS WORK

It is known that human activity in an environment causes interference in radio signals. There are several ways to use the CSI for activity recognition [1]. Early on, received signal strength indicator (RSSI) was popular as a feature for activity recognition, but it is not as refined as the CSI that is commonly used now [5]. The DL approach to human activity detection and recognition shows great promise, in [1] several techniques that already have been successfully applied to the task are presented. Some DL methods (e.g., recurrent neural networks, RNN) focus on using the temporal patterns in the data [4], while other methods (e.g., convolutional neural networks, CNN) are suited for spatio-temporal patterns [6]. A downside of the DL approach is that it may require time and resources to be spent on data collection and labelling, and on training the algorithms [7], [8]. On the other hand, DL requires less attention on manual feature design than conventional ML methods.

Simpler neural network architectures (e.g. multi-layer perceptron, MLP) show potential at detection, localisation and recognition tasks [1]. CSI amplitude and phase can be used as features in MLPs. However, preprocessing the phase information may take some effort, especially if using commercial WiFi equipment, since the carrier frequency offset may render the phase estimates too noisy [9]. In [10], a linear transformation was used to process phase data used in an indoor localisation task, but the method was not applicable in the case presented in [9]. Instead, in [9] the phase shift of the signal paths was estimated by inferring the path length change from the channel frequency response (CFR) power.

Ding and Wang [4] proposed a twostep detector that used a decision tree that takes sub-carrier variance and correlation coefficient from raw CSI data to detect human activities. An RNN was used to classify the activity type. The detector needed to be retrained to be successfully deployed in a new environment.

Generalising detection methods to different indoor environments and deploying the system outdoors are challenging tasks. Many studies focus on using WiFi-signals, as the techniques may be deployed in commercially available (and ubiquitously present) hardware platforms [1]. WiFi is limited in outdoor environments, making it unfeasible for long-range detection. Experiments with Long Range (LoRa) signals successfully located a person in a larger area (localisation error was less than 4.6 meters in 90% of cases) using a single transceiver pair [11]. This study will focus on activity recognition in indoor environments using signals in different frequency bands.

Minor changes in the (static) environment may have a significant impact on signal propagation [5], [7]. To overcome the need to retrain the system every time it is deployed in a new indoor environment (also called domain adaption), autoencoders (AE) can be deployed [1]. In [5], AEs were used to generalise CSI features used for indoor localisation. A CSI profile was applied in a reference environment and an algorithm was implemented for detecting a new static environment. They established a CSI profile in a reference environment and implemented an algorithm for detecting a new static environment. Functions for transforming the new environment features into the reference environment features were estimated and applied to data if a new environment was detected. These features were given as input to an AE. The AE was trained to reproduce and emphasise critical features (CSI features relating to the new environment are suppressed), which boosted localisation performance. In [9], environment independent motion speed measurements were leveraged to achieve 80% recognition accuracy in a domain that the model had not been trained in. The key was to realise that the speed of the path length change is similar for the same activity regardless of the static environment. This study will investigate the domain adaption capability of different methods.

Conventional (shallow) ML methods, including support vector machines (SVM), random forests, and k-nearest neighbours (kNN), have been successfully deployed at small scale activity recognition tasks [1]. The feature quality is important for shallow learning methods, and performance can be boosted significantly by using manually designed features compared to using raw or preprocessed CSI data [7]. Features can be designed through domain knowledge or applying dimension reduction techniques, such as principal component analysis (PCA), independent component analysis (ICA), or singular value decomposition (SVD). The unsupervised learning method k-means clustering can be applied for feature verification [12]. In [9], one hidden Markov model (HMM) was created for each activity, the likelihood of the observations (feature vectors) were used, picking the activity that has the highest likelihood of producing the data.

3 SYSTEM DESCRIPTION

This section provides the reader with an overview of the components of the system, and how they interact to achieve the system's purpose. The complete set of system requirements can be found in [13].

3.1 Overview

The purpose of the system is to detect dynamic events in indoor environments by using ML algorithms to analyse signals from Software-Defined Radio (SDR) equipment. While SDR has no unique definition, it implies that the sent waveform can be modified by the software without changing the underlying hardware or environment [14]. This flexibility allows the system to optimise detection in different environments. The detector is built with a multiple antenna setup consisting of four Pluto SDR transceivers connected to a host computer. One device is a transmitter (Tx) and three devices are receivers (Rx). The system consists of three subsystems, hardware, software, and user interface, all of which were implemented using Python. The hardware subsystem consists of four Pluto SDR devices and is responsible for collecting the data and store it in the right place. This is done by performing a channel estimation on the received signal given the known transmitted signal. Functionality to send Binary Phase Shift Keying (BPSK) symbols for evaluation of the channel has been implemented in software but is not integrated with the rest of the system due to unreliable hardware, this topic is discussed further in Section 4. The data is collected and each observation is labelled according to the class it represents. The hardware subsystem is presented in Section 4. The collected data is used by the software subsystem to train supervised ML algorithms and one deep neural network (DNN) model. Before the models are trained the data is preprocessed and features are extracted. Then, seven different types of models are built; SVM, HMM, k-means, decision tree, random forest, Gaussian mixture model (GMM) and DNN. Classifications are made with test data to evaluate the models, but also new non-labelled observations can be used for prediction using the implemented live classification functionality. The software subsystem is presented in Section 5. The third subsystem is the Graphical User Interface (GUI). The GUI can be used to collect data, train models, and make predictions. The GUI is presented in Section 6. The detector is designed with the aim to make (accurate[1]) classification decisions in binary and multiple hypothesis scenarios. The different subsystems and their dependencies are shown in Figure 1.

The following classes are used for the classification. Balloon, walking, dancing and jumping are considered as dynamic classes while static is the only static class.

- **Static** – The room is completely empty while collecting data.
- **Balloon** – One person in the room, waving one balloon back and forth in different speeds in the area between the receivers.
- **Walking** – One person walking back and forth in the room, mostly in the area between the receivers but also little bit on the side.
- **Dancing** – One person dancing (moving fast, with different parts of the body) in the area between the receivers. Trying to do movements as different as possible from walking and jumping.
- **Jumping** – One person standing in the area between the receivers jumping up and down, changing place now and then.

1 see the requirement specification [13].

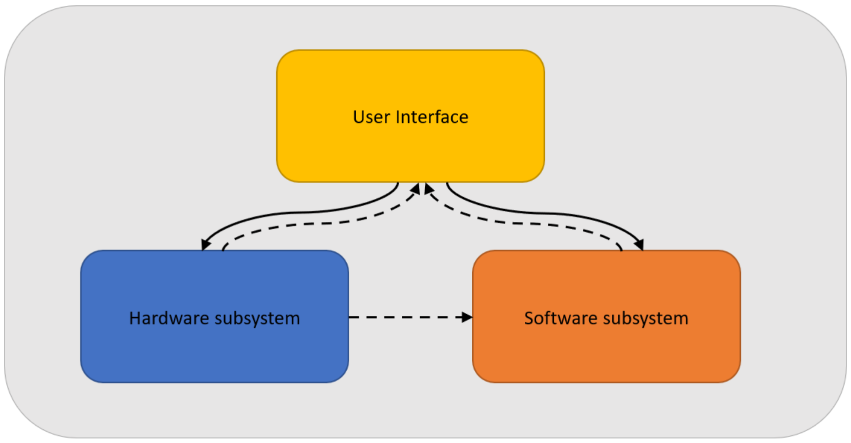


Figure 1: System overview. Solid lines indicate control signals, dashed lines indicate data flow.

3.2 Data Collection

The data collection includes getting training and test data for the classification algorithms. During one data collection session, the static environment and the chosen dynamic activities described in Section 3.1 are performed and collected for a fixed number of minutes that the user decides.

A figure describing the data collection can be seen in Figure 3. Each time an activity is performed, it is recorded for the fixed amount of time, during which M packets are collected. The channel is assumed to be time invariant during the transmission of one packet and the channel is estimated during the transmission of each packet. Each channel estimate is calculated based on a packet of size N samples which corresponds to the parameter binsize.

A data collection setup is shown in Figure 2. The collected data is sent to the software subsystem for further processing which can be seen in Figure 1. The final data set that the models are trained with have 10 minutes of data collected for each class, with different people performing the activities within each class.

An experiment with collecting data and performing live classification was also performed by using the trained models and get the result by a voting between them while collecting live data. The result for the live classification is presented in Section 7.6.

3.2.1 *Environment*

The data is collected in an empty, small room. An approximate room sketch is given in Figure 4. The signals are transmitted from one Pluto SDR device. Three receivers are used, see Section 4.4 for more information about multiple receivers. The receivers are placed approximately 1.5 meters from the transmitter, and it is approximately 1 meter between the receivers. If the devices would be moved to a new environment, then it would be necessary to collect new training data and to retrain the models.

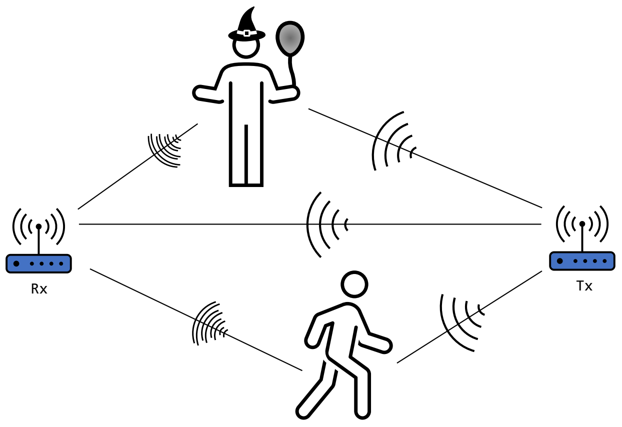


Figure 2: A data collection scenario for a dynamic event. One (or several) persons or objects perform some activity in the room. A signal is transmitted from the one Pluto SDR device (Tx) to the other (Rx).

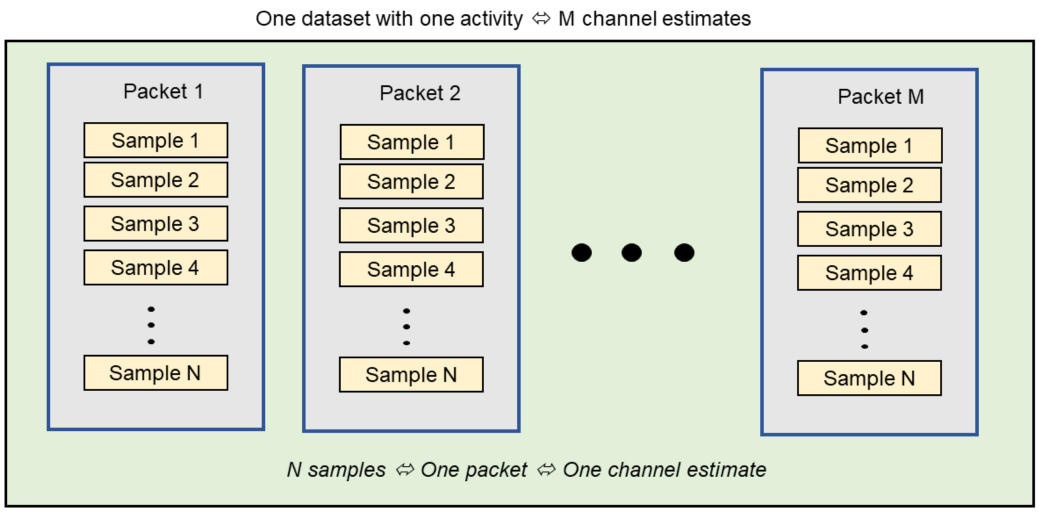


Figure 3: Description of one data collection for one activity. After collecting N samples (one packet), a channel estimate is performed. This is repeated until the activity stops after a predetermined time.

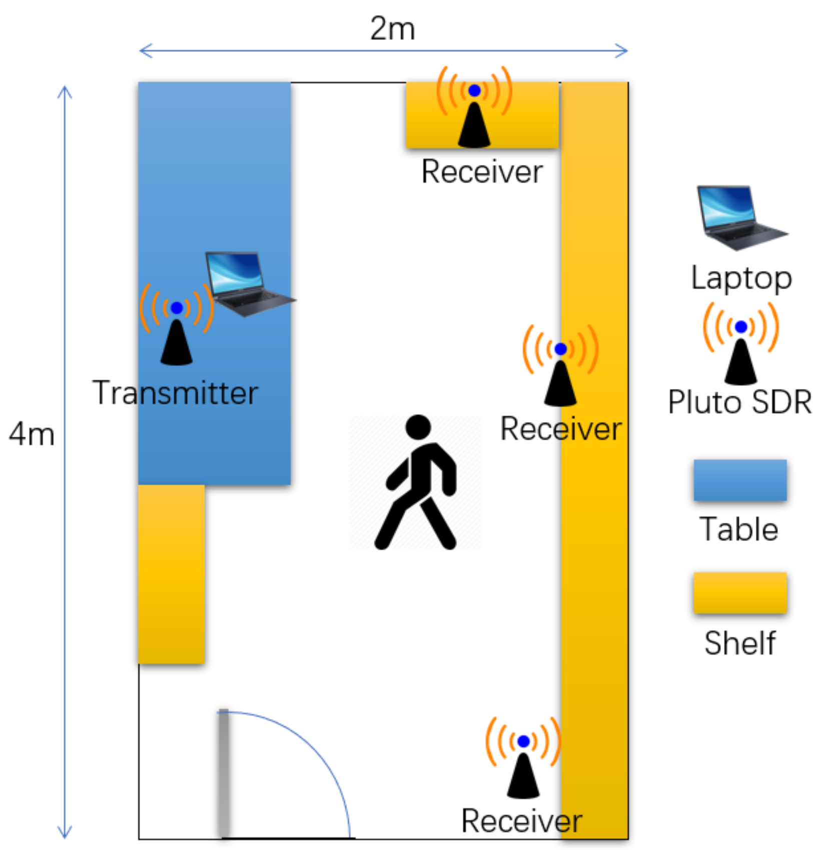


Figure 4: Data collection environment sketch. The four Pluto SDR devices operate as the transmitter and receivers.

4 HARDWARE

The hardware subsystem consists of Pluto SDR devices with specifications according to [15]. These devices are connected to a host computer that handles their transmit and receive functionality. The software used to calibrate the Pluto SDR devices and to estimate the channels between the transmitter and receivers is considered to be a part of the hardware subsystem. The output of the hardware subsystem is the estimated CSI, to be further processed by the software subsystem, as well as a label of which environment the data was collected in.

The Pluto SDR devices make it possible to abstract away the analog signal processing and it is thus sufficient to consider the data transmission as a discrete memory-less channel that takes IQ symbols (complex numbers, where the real and imaginary parts are called in-phase (I) and quadrature (Q) components, respectively) as input and gives distorted IQ symbols as output. The Pluto SDR devices are fixed in place in the indoor environment during each data collection session, which means that the channel can be assumed to be time invariant in the static environment and that it will be time variant when the environment is dynamic.

During the data collection phase, appropriate symbols known by both transmitter and receiver will be sent and based on the corresponding received symbols, the channel can be estimated at the receiver side. This is repeated until a sufficient amount of CSI data has been gathered. Each of these channel estimates will be assigned a label specifying the type of event that occurred during the corresponding data collection.

The multiprocessing package in Python was used to send and receive using the same script. It made it possible to run the transmitter and the three receivers simultaneously on the same host computer by sharing the computer's resources between the devices.

4.1 Channel Estimation

As stated earlier, the analog signal propagation can be abstracted away when using the Pluto SDR devices. It will be assumed that the transmission is divided into time-frequency parts that fit into a coherence interval, which is the time-frequency interval during which the channel is time invariant, and its frequency response is constant. The single-input single-output (SISO) transmission of a packet of N (also referred to as binsize) IQ symbols, where N channel uses are assumed to fit into a coherence interval, can be modelled as

$$\mathbf{y} = \sqrt{\rho} h \mathbf{x} + \mathbf{w}, \tag{1}$$

where $\mathbf{y}$, $\mathbf{x}$ and $\mathbf{w}$ are the $1 \times N$ vectors of received symbols, transmitted symbols (assumed to be known) and independent and identically distributed (i.i.d.) complex Gaussian distributed random variables with zero mean and unit variance ($\mathcal{CN}(0,1)$) noise entries, $h \in \mathbb{C}$ is the channel (assumed to be a random variable with a constant value during each packet of N channel uses) and ρ is the signal-to-noise ratio (SNR). The system implemented in this project uses $\mathbf{x} = \frac{1}{\sqrt{N}}[1, 1, ..., 1]$ and the channel can then be estimated with the maximum likelihood estimator

$$\hat{h} = \underset{h}{\text{argmax}}\{p(\mathbf{y}|h)\} = \underset{h}{\text{argmax}}\{p(\mathbf{yx}^H|h)\} = \underset{h}{\text{argmin}}\{\|\mathbf{yx}^H - \sqrt{\rho}h\|^2\} = \frac{\mathbf{yx}^H}{\sqrt{\rho}} = \frac{1}{\sqrt{\rho N}}\sum \mathbf{y}, \tag{2}$$

where $p(\mathbf{y}|h) \sim CN(\sqrt{\rho}h\mathbf{x}, \mathbf{I}_N)$ and $p(\mathbf{yx}^H|h) \sim CN(\sqrt{\rho}h, 1)$ are the probability density functions of $\mathbf{y}$ and $\mathbf{yx}^H$, respectively. A single-input multiple-output (SIMO) antenna setup with three receivers was used in this project. The channel between each pair of transmitter and receiver is then seen as a SISO channel and the channels can be estimated independently of each other by calculating (2) at each receiver in parallel. Since there was no need to find the actual channel, the SNR was assumed to be 1 when calculating $\hat{h}$. This means that a scaling factor is missing, but it is of no interest because the software subsystem performs a normalisation that would cancel out the effect of using the actual SNR value anyways.

4.2 Evaluation of Channel Estimate and Offset Compensation

The channel estimates can be evaluated by sending BPSK symbols when using the estimated channel in the decoding and calculating the bit error rate (BER). Sending BPSK symbols means that $\mathbf{x}$ in (1) only consists of the IQ symbols 1 and -1, i.e., one symbol corresponds to one bit. Hence, calculating the uncoded BER is equivalent to calculating the proportion of symbols that were not decoded correctly. Since the SNR is not known in advance, it cannot be predetermined what a good BER is. Thus, the BER can only be used as a comparison between different channel estimates.

Using the channel in practice shows that BER does not only depend on the transmission channel, but it is also affected by the Pluto SDR devices' offset. Due to aging, temperature or voltage conditions there is a frequency drift in the Pluto SDR devices up to ± 25 ppm [15]. Sending and receiving on two different devices implies that the frequency offset can be as large as 50 ppm, which translates to $915 \cdot 10^6 \cdot 50 \cdot 10^{-6} \approx 46$ kHz when using the carrier frequency 915 MHz. This offset needs to be compensated if the transmitted symbols are to be recovered at the receiver. Orthogonal frequency-division multiplexing (OFDM) is used in the data transmission to provide better reliability for calculating the offsets.

Time, frequency, and phase offsets can be found by inserting a known pilot into the transmitted symbol sequence. The time offset is detected by calculating the correlation between the received signal and the pilot at the receiver. The frequency offset is detected by calculating the average of symbol-timing offset between neighbouring symbols. Theoretically, the symbol-timing offset increases at the same rate due to the mismatch in carrier spacing. The phase offset is detected by calculating the offset between the received pilot and the original pilot, which is used to compensate the received symbol sequence to restore the transmitted symbols at the receiver.

4.3 Setting Parameters

There are some parameters that are possible to set for the hardware. These parameters are center frequency, sample rate and binsize. Binsize is the number of samples used for one channel estimation (N in (2)). It is possible to set center frequency and sample rate from the GUI, but a default value for each parameter is provided. The default values of the parameters have been set by testing. The user cannot set the binsize in the GUI, but this parameter has also been set by the same test. Different values of center frequency, sample rates and binsizes have been tested. The different values tested for each parameter were

- Center frequency: [815 MHz, 915 MHz, 1015 MHz]
- Sample rate: [0.55 MHz, 1 MHz, 2 MHz]
- Binsize: [10, 100, 250]

4.4 Multiple Receivers

The group initially had two Pluto SDR devices available, but during the project two additional devices were accessible. Different ways of exploiting multiple antennas were investigated. It was decided to use a SIMO setup with three receivers, since using several transmitters would have caused difficulties with time synchronisation of sent and received symbols. The transmitter would also have needed to send orthogonal symbol sequences. By using only one transmitter, and by sending a constant signal (in the baseband), it was possible to estimate the channels at each receiver independently of the others. Different methods of exploiting the additional information was investigated and is explained in Section 5.

5 SOFTWARE

5.1 Feature Extraction

The feature extraction pipeline implements a method for transforming the raw channel estimates collected by the hardware subsystem into features using a custom set of transformation functions. The collected data may be split into samples of different length (e.g., 3 or 10 seconds) before applying the chosen functions, which include the mean and variance of the input absolute values. The pipeline also creates a set of time series features to be used by the HMM and DNN. Each feature is calculated for every channel unless stated otherwise. A description for each of the implemented features now follows.

Amplitude mean: The mean amplitude of the channel estimates in one observation.
Amplitude variance: The amplitude variance of the channel estimates in one observation.
Amplitude max difference: The maximum difference between two amplitude values in one observation.

Channel correlation: The correlation coefficient for the amplitude of the channel estimates between each pair of channels. Each series of channel estimates is treated as one variable.
Phase mean: The mean phase of the channel estimates in one observation.
Phase variance: The phase variance of the channel estimates in one observation.

A static offset that changes randomly between sessions was observed in the received signal amplitude. To get a more representative result for the amplitude mean, a calibration routine was developed. At each new data collection the environment is left in a static state for a short while, the mean amplitude is calculated and this value is then removed from the data collected in the current session. This procedure removes irregularities introduced by the hardware in an appropriate way if the hardware is the source to the irregularities. However, if the irregularities naturally arise from the environment this calibration should not be done. If the irregularities come from the environment it is possible that the amplitude mean is not a good feature because it is too sensitive to changes in the environment.

5.1.1 *Outlier Removal*

For certain models, like the HMM, it was initially found that with a limited amount of data, the model would be negatively affected by outliers. To mitigate this problem, a method for outlier removal was explored and implemented in the feature extraction step using LocalOutlierFactor from sklearn. The procedure removes a given proportion of data points that are considered to be outliers based on the local density of a given number of its closest neighbours. The usage of local density makes the method robust to varying cluster densities.

5.1.2 *Normalisation*

The extracted features may be normalised in the feature extraction pipeline, which may help to balance the importance of features that take values in different ranges. Min-max normalisation is used to map each feature onto the range $[0, 1]$ by the transformation

$$\mathbf{x}^* = \frac{\mathbf{x} - \mathbf{x}_{\text{min}}}{\mathbf{x}_{\text{max}} - \mathbf{x}_{\text{min}}},$$

where $\mathbf{x} \in \mathbb{R}^N$ is the vector containing a certain feature from all samples, and $\mathbf{x}_{\text{min}}$ and $\mathbf{x}_{\text{max}}$ are the smallest and largest values of $\mathbf{x}$ respectively.

5.1.3 *Time Series Data*

For the data that is used by the HMM (and DNN), the vector of channel estimates is broken into a time series of T points. The features (e.g., mean amplitude) is thus computed from a portion of all channel estimates. The features are then (optionally) normalised by the min-max method, so that the minimum value for each feature (over all time points) is zero, and the maximum value is 1. Data from different channels are treated as separate features. The whole time series is treated as one feature in the normalisation step, so that the features in each time point are not normalised separately.

5.1.4 *Noise Reduction*

A method for noise reduction was implemented and used before the calculation of the features based on the amplitude. The implemented method was a Butterworth lowpass filter. Adding the lowpass filter on the features based on amplitude improved the results for the models.

5.2 Machine Learning Models

The ML models used for classification are presented in this section. These models were chosen for being widely used and accepted models that perform good in many applications and therefore could be assumed to perform good in this project as well. The HMM was used in [9] and was therefore implemented but using different features.

5.2.1 *K-means Clustering*

The k-means clustering algorithm is an unsupervised method for structuring a set of unlabelled data into groups based on a given measure of similarity [16, Chapter 13]. K-means clustering begins by initialising k cluster centers randomly in the feature space and assigns each sample to the closest center. The cluster center is updated to be the mean of all data points assigned to the cluster, and the algorithm iteratively performs these computations and updates until convergence. In this way, the sum of residual squares with respect to the similarity measure is minimised [17, Chapter 9]. The algorithm may also be used when class labels are available. One method is to apply the algorithm for each class separately to obtain several cluster centers per class, which in turn are used in prediction in the usual way [16]. Another is to take the true class means directly. In our implementation, the former method was used.

The performance of k-means is sensitive to outliers (large distances gives a large addition to the residual sum of squares) and in-class variance. Using the supervised version described above may alleviate the impact of these factors, since true labels are available, and the cluster centers can adapt better to the different shapes of the classes. Outlier removal was discussed in general in the previous section.

The advantage of k-means clustering is that it is easy to interpret, has few parameters, and is comparatively less demanding with respect to computations. However, since this setting has only a handful of features, and the data set is relatively small, the only advantage that is useful is the interpretability of the model.

5.2.2 *Gaussian Mixture Model*

The Gaussian mixture model (GMM) is a generalisation of the k-means algorithm that describes the data as being generated by a set of k Gaussian distributions with mean, μ_i, and covariance, Σ_i, $i = 1, ..., k$ [17, Chapter 9]. In contrast to k-means, GMM is a soft classifier, and therefore assigns a data point to a cluster with a probability, and not a hard assignment. The model parameters are found through the expectation maximisation algorithm which iteratively computes the posterior of the data, and maximises the likelihood of the data with respect to the parameters [17]. GMM reduces to k-means as $\Sigma_i = \varepsilon I, \forall i$, and $\varepsilon \to 0$. GMM has some similarity to the HMM, in that a mixture of Gaussians is assumed, but it does not utilise the temporal information in the data.

The GMM was implemented in the same way as described for k-means above. The algorithm was applied once for each activity class, and the found parameters were merged into one model. Since it is a generalisation of k-means it is expected that the model would perform at least as good in the classification.

5.2.3 *Support Vector Machines*

One of the models implemented to do classification is the SVM. To implement it, the class sklearn.svm.SVC from sklearn was used. For the multi-class case it uses the one-vs-one approach. This produces one classifier for each pair of classes in the training data. Using the GridSearchCV function from sklearn, different parameters and kernels have been tested. The considered kernels are a linear kernel (4)

$$K(x, x') = x^T x', \tag{3}$$

a polynomial kernel

$$K(x, x') = x^T x', \tag{4}$$

and a radial basis function kernel

$$K(x, x') = \exp(-\gamma \|x - x'\|^2). \tag{5}$$

Apart from selecting kernels the values for the parameter γ and penalty parameter C must be selected. The penalty parameter reflect how much error that is bearable, large value on the parameter C leads to an overfitted model whereas a low value gives a smoother boundary [16, Chapter 12]. The parameter γ in the radial basis kernel controls how much influence each training point is given in the final model. A low value gives influence also far away from the training points while a larger value results in a model where training points only influence the model close to themselves. The combination of kernel function and parameters that give the highest accuracy is considered to be the best model.

One of the benefits with the SVM model is that each point to classify does not need the kernel function to be evaluated at each training point. Instead, it is only evaluated at a subset of points, at the so called support vectors [17, Chapter 7]. If the classes are linearly separable in the feature space, good results can be assumed to be achieved by using the linear kernel. However, if the data cannot be linearly separated, a polynomial or radial basis-kernel will probably perform better.

5.2.4 *Decision Trees*

Another method that is easily extendable for classification of multiple classes have been implemented: the simple (yet powerful and transparent) decision tree [18, Chapter 12]. Due to the transparency of the model, it is easy to understand how the decisions are made by the decision tree. A drawback is that they seldom are very accurate [16, Chapter 10]. The model is a tree-structure with binary decisions at each node. At the leaves of the tree the class assigned is specified. An advantage of using this model is the prediction speed since it is on the order of $\log(n)$ with respect to the depth of the tree.

The split at each node is done with the candidate split θ, including both the feature and threshold for the split, that solves

$$\theta^* = \text{argmin}_\theta G(Q_m, \theta), \tag{6}$$

where the function to minimize is

$$G(Q_m, \theta) = \frac{N_n^{\text{left}}}{N_m} H(Q_m^{\text{left}}(\theta)) + \frac{N_n^{\text{right}}}{N_m} H(Q_m^{\text{right}}(\theta)). \tag{7}$$

Q_m represents the data at node m, N_m the number of samples at the node, and $H(\cdot)$ is an impurity function that measures of how good the classifier performs at that node. For the impurity function the Gini measure is used,

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}). \tag{8}$$

The probability p_{mk} is defined

$$p_{mk} = \frac{1}{N_m} \sum_{y \in Q_m} I(y = k) \tag{9}$$

and represents the proportion of observations at the node belonging to class k. $I(y = k)$ is an indicator function that evaluates to one if the the class of y is equal to the class k, and otherwise to zero. The Gini measure is a measure of how often an element in the data set at the node would be incorrectly classified if it was classified randomly according to the distribution of the data Q_m at node m.

The depth of the tree and the number of terminal nodes are parameters that either must be selected when the model is trained or controlled by post-pruning the tree. Making the tree too deep can easily lead to overfitting. The approach used in this project was minimal cost-complexity pruning. The method helps to select the best model by considering the trade-off between accuracy and model refinement. The post-pruning algorithm starts with the trained tree and removes nodes step by step, then the tree with highest accuracy for the validation data is selected as the final model.

5.2.5 ***Random Forest***

This model builds on the structure of decision trees and uses a variant of bootstrap aggregation to reduce the variance of the prediction function [16, Chapter 15]. The principle for building a random forest model is to first draw a bootstrapped set of samples from the training data. All samples have equal probability to be selected each time and duplicates are allowed to be drawn. Then a tree is fitted for the bootstrapped data, but at each split only a subset of variables is considered and the best split among them is used. Compare this to the decision tree where all features are considered at each step. The process is performed until the desired number of trees have been created. Classification is made by majority voting by the trees.

It can be suspected that a fitted random forest will perform better than a simple tree for us, as the method will result in more different features being considered at each step in the tree fitting process.

5.2.6 ***Hidden Markov Model***

The HMM is a model for inference on time series data. An HMM describes a series of transitions between states $Z = \{Z_t^0, ..., Z_t^q\}$ in a Markov chain, which cannot be observed directly, but that stochastically control a set of observable variables $\mathbf{X}^t = \{X_0^t, ..., X_p^t\}$, also called emissions. The model is completely described by three sets of parameters: initial probabilities for each hidden state $p(Z_0^i)$, a set of transition probabilities between states $p(Z_{t+1}^i|Z_t^j)$, and a set of emission densities $p(\mathbf{X}_t|Z_t^i)$, which can be, for example, multinomial or Gaussian. The emissions in a time step t depend only on the hidden state in that time step.

Inspired by Wang et al. [9], the system implements predictions on activities in the environment by considering the features as observations, and the hidden states as the unknown factors that influence the channel. Each activity does not correspond to a hidden state, but rather to a set of hidden states in an HMM. One HMM with Gaussian emissions was trained for each activity and the prediction of the system is the activity corresponding to the model that had the highest likelihood of generating the chain of observations in the sample. The underlying assumption is that each activity can affect the channel response in different ways, and that the observations are normally distributed given the hidden state. The latter might seem like a strong assumption, but plots of the data show that it is not unreasonable as a model. The strength of the HMM is that it models observations over time, and may be able to extract more information than, for instance, k-means clustering.

The model is fitted by the expectation maximisation (EM) algorithm, and when predicting, the likelihood of the data is computed by the forward-backward algorithm (a single forward pass) [17, Chapter 13]. The EM algorithm is not guaranteed to find a global optimum. The HMM must infer both a set of transition probabilities and emission probabilities. The number of parameters is determined by the number of hidden states, which is a design choice. A specific interpretation of the hidden states is hard to find, but in theory they should relate to different modes of the environment (variations of an activity in this case). Setting a high number of states introduces more parameters to estimate, which will require a larger amount of data. The experiments in this study were limited by the amount of data, see Section 7.5.1 and the discussion in Section 8.4.1. The HMM framework uses the hmmlearn package which was orphaned from sklearn and is now a separate repository [19].

5.3 Deep Neural Networks

A DNN model was also used to classify the data, more specifically an MLP classifier. An MLP is a neural network with multiple layers of neurons. It consists of an input and output layer, with hidden layers in between.

A neuron in the multi-layer perceptron is an algorithm that takes inputs $x_1, ..., x_m$ and multiplies them with weights $w_1, ..., w_m$, sums them together and adds a bias b. The result is passed through an activation function (in this case $\mathrm{ReLU}(x) = \max(0, x)$), which gives the output $\phi = \mathrm{ReLU}(b + \sum_{i=1}^{m} x_i w_i)$. Figure 5 shows an example of how a perceptron works.

In the DNN there is an activation function, which is the second step of the perceptron classification. The purpose of the activation function is to decide whether a neuron should be activated or not. The activation function is a transformation that is done over the input that delivers an output based on that. In this project, for the MLP, a rectified linear activation (ReLU) function was used. The main advantage of the ReLU function is that is does not activate all the neurons at the same time. It does this by converting negative input to zero, and therefore the neurons with zero input will not be activated. This is the most common activation function and is less susceptible to vanishing gradients than the sigmoid and tanh activation functions. It was chosen due to this, and because it is the default recommendation in modern neural networks [20].

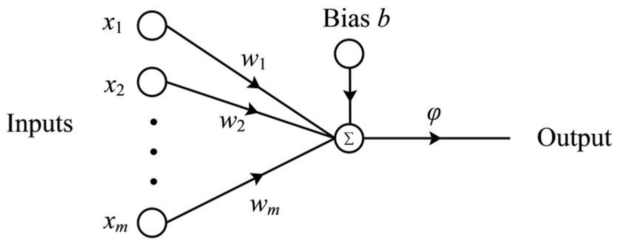


Figure 5: Image of how a neuron calculates the weighted sum of the inputs.

The training of the neural network happens over several iterations, which is split into batches. The sklearn neural network chose the batch size as $\min(200, \text{n\_samples})$. With our samples being over 200, a batch size of 200 was chosen. We did not test other batch sizes, but it is something that can be done. The training was run for 1000 iterations and used backpropagation.

The library used for the neural network was from Python sklearn, the MLPClassifier, which is as supervised learning algorithm. The MLPClassifier was chosen because it was easy to implement, but still did what it needed to. DNN via Keras was also tested, but there was no advantage with that over the sklearn classifier, and because it was more difficult to implement, it was excluded.

The MLP was constructed with four hidden layers, each with 512 nodes. The number of hidden layers and nodes was chosen in regard to the number of features, and also by testing and evaluating the performance of the classifier.

5.4 Training and Testing the System

The processed data containing observations with features and labels are divided into training, validation and test data sets which are used for implementation and testing. The training data is used to train the models. The validation data is used to tune parameters in some of the models by training them with training data and then try different parameters in the model and evaluate the result with the validation data. Test data will be used to evaluate the result for the final models. For comparison, the same data will be used on all models.

The Python package sklearn contains functions that were used to tune parameters for models. These functions can, for example, evaluate different specified options for parameters using scores of user's choice by cross validation. For this, the training and validation data is used. The function returns the final model.

5.5 Evaluation of Classifications

Different kinds of evaluation methods and measurements are used to evaluate the models. True positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) are defined for binary classification problems. The definitions are shown in Table 1. Positive and negative are the classifications made, and true and false stands for if the classification was done correctly.

Table 1: Definitions of TP, TN, FP and FN samples respectively.

Predicted \ Actual	Positive	Negative
Positive	TP	FP
Negative	FN	TN

This kind of table, where the result from a classification is presented, is called confusion matrix, and gives an overview on result from the classification. By using these definitions, it is possible to define measures that gives an overview on how well a classification performed on a full data set. Test data is always used to evaluate a classification model. Accuracy is the rate of correctly classified samples and can be expressed as

$$\text{Accuracy} = \frac{\text{Correctly classified predictions}}{\text{Total number of predictions}} = \frac{\text{TP}+\text{FN}}{\text{TP}+\text{TN}+\text{FN}+\text{FP}}. \tag{10}$$

Precision can be interpreted as the rate of how many classifications that was correct for one class in relationship to the number of samples for this class. Precision can be expressed as

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}. \tag{11}$$

Recall is the rate of how many classifications that was correct for one class in relationship to how many samples that got classified as this class. Recall can be expressed as

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}. \tag{12}$$

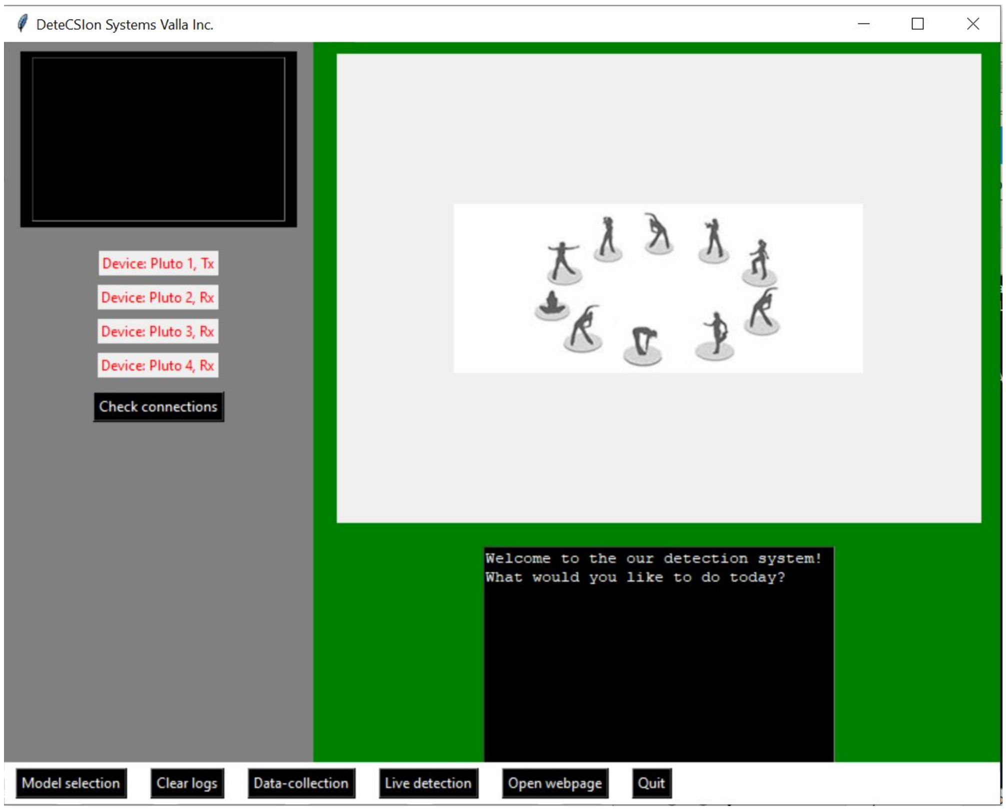
The harmonic mean between precision and recall is called F1-score, which can be expressed as

$$\text{F1-score} = 2 \cdot \frac{\text{Recall} \cdot \text{Precision}}{\text{Recall}+\text{Precision}}. \tag{13}$$

Another evaluation measurement used is the number of seconds of data the model needs to make a classification, which corresponds to the number of seconds for one observation.

6 GRAPHICAL USER INTERFACE

One subsystem of the final product is the GUI, through which the user can control the whole system. Figure 6 shows the front page of the GUI. The GUI was created using Python and the Tkinter package. The user can initiate data collection, train a model, start a live-session, and access the project website from the GUI. The optimal parameters



```
Welcome to the our detection system!
What would you like to do today?
```

Figure 6: The front page of the GUI.

presented in Section 7.2 was used as default values in the GUI, but customisation is allowed. The GUI was designed to be generalised and adaptable to the user.

A detailed guide on how to use the GUI can be found in the User Manual [21].

7 RESULTS

The most important results in the project are presented in this section.

7.1 BPSK

As stated in Section 4.2, the hardware device offset needs to be compensated to show the channel effect on the transmitted symbols. By inserting a known pilot to the transmitted symbol sequence, time offset can be detected by

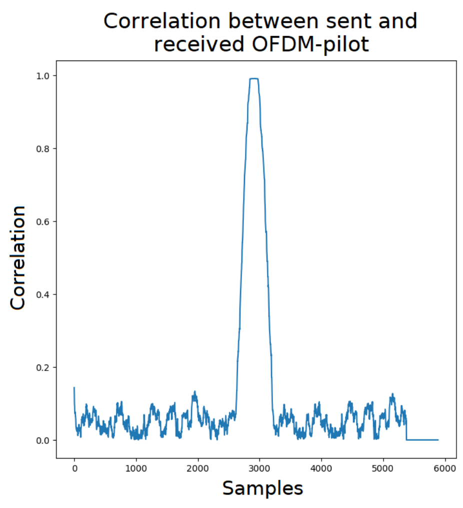


Figure 7: Pilot auto correlation in time offset detection. The x-axis shows the offset in samples.

calculating the correlation between the received signal and the pilot at at the receiver. Figure 7 shows a correlation result. Frequency offset and phase offset are as stated in Section 4.2. After calculating all the offsets, the received symbols can be compensated. One example result, where 1024 data bits were sent in a static environment, can be seen in Figure 8, where all the symbols are clustered close to -1 and 1 with acceptable phase shifts. The BER is zero in this case.

7.2 Hardware Parameters

As stated in Section 4.3, a test for finding the best values for the default hardware parameters was performed. Different hardware settings were used, and classifications were done with the different models and different lengths of time per observations. The result is presented in Figure 9 and Figure 10. The box with the accuracy for each model and setting is coloured green if the accuracy is above 80% which makes it possible to see that the column with most green boxes is data set 10 and data set 11. Data set 11 is chosen since the binsize in this data set was larger. The final hardware parameters that performed best in this test and was chosen to be default values was the following.

- Center frequency: 915 MHz
- Sample rate: 550 kHz
- Binsize: 100 samples

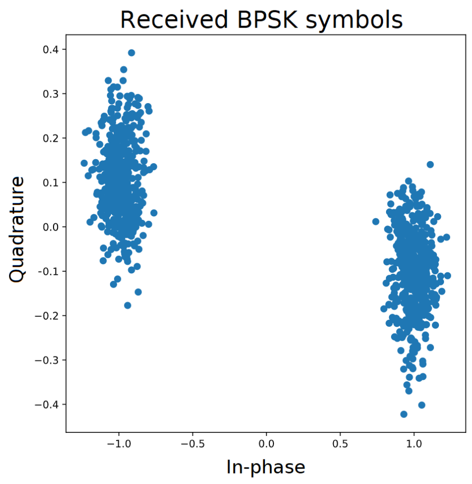


Figure 8: Received BPSK symbols. As desired there are clear clusters for both sent symbols, 1 and -1.

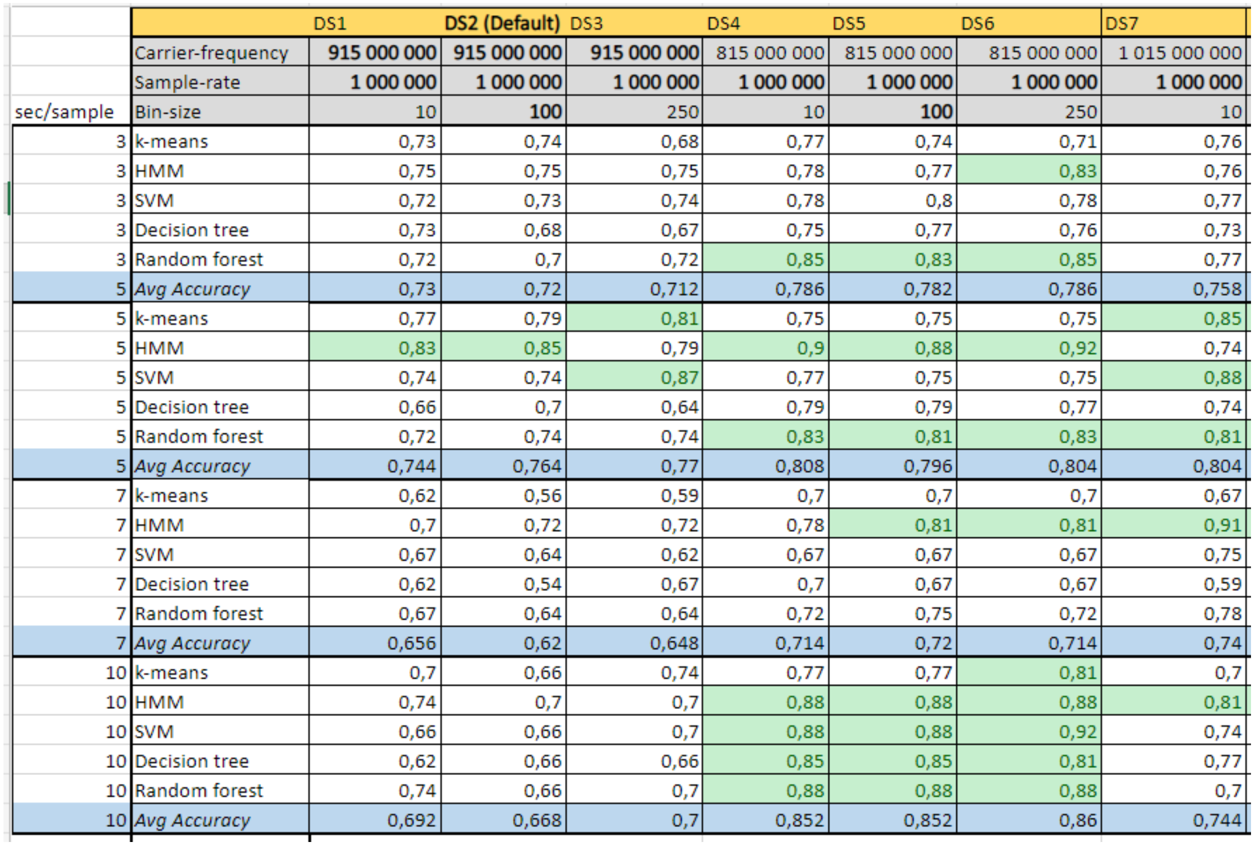
7.3 Collecting Data

The data was collected in batches of one minute per collection and was then split up into multiple observations, all labelled to be from the same class. Figure 11 shows a plot of one CSI sample collected in a static environment and Figure 12 shows one CSI sample collected while waving a balloon back and forth in the room.

7.4 Features

In Figure 13 four plots representative for the feature distribution can be seen. Two-dimensional plots were selected to facilitate readability, but also three-dimensional plots were analysed. It is hard to draw clear conclusions from only visual inspection, but it does not seem infeasible that amplitude variance together with the amplitude max difference could give a clean split between the static class and the dynamic classes when data from all three channels is used. Most probably using these two or just one of them would give good results in the binary hypothesis testing between static and dynamic. The channel correlation, phase mean, and phase variance features do not seem to give clear divisions for any classes, neither together nor one by one. A somewhat better division was achieved for one of the channels, but the overall impression is that these features are less good than the other three. Figure 14 shows the three features considered as the best ones plotted against each other in 3D. Clustering of data for some the classes can be seen, and it can be suspected to be even better with the same features for additional channels.

In Table 2 the average accuracy for all classes and average F1-score for the static class is summarised for all models given different sets of features. First, all features were used individually to see both how the models trained with them perform on all classes and how the models perform on only the static class by looking at the accuracy and F1-score respectively (feature set 1-6). Then the features that did best at distinguishing static events from dynamic were used together (feature set 7). Followed by using all features together (feature set 8). After that it was tested to exclude one feature at a time from the less effective ones to see whether it would improve or reduce the performance (feature

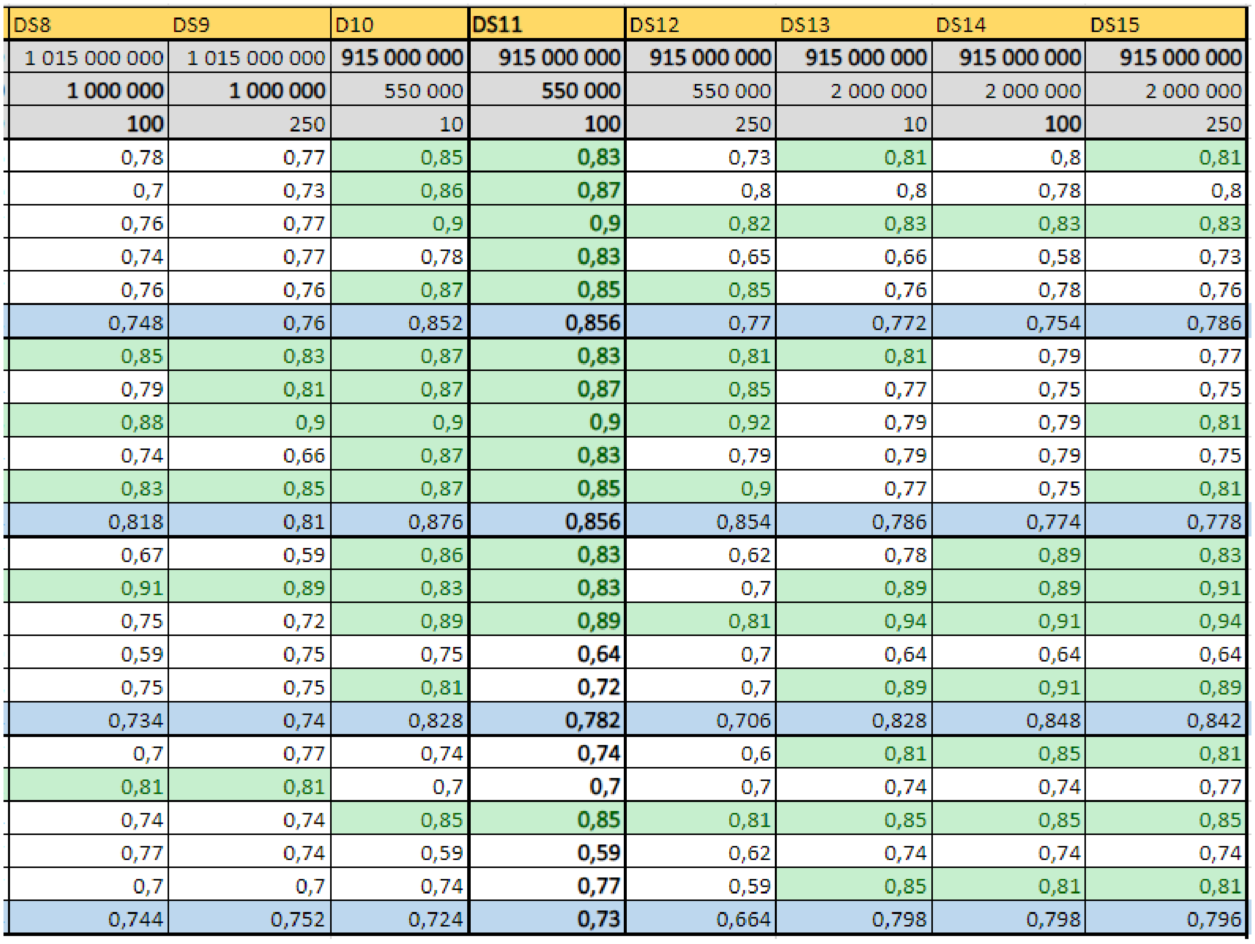


		DS1	**DS2 (Default)**	DS3	DS4	DS5	DS6	DS7
	Carrier-frequency	**915 000 000**	**915 000 000**	**915 000 000**	815 000 000	815 000 000	815 000 000	1 015 000 000
	Sample-rate	**1 000 000**	**1 000 000**	**1 000 000**	**1 000 000**	**1 000 000**	**1 000 000**	**1 000 000**
sec/sample	Bin-size	10	**100**	250	10	**100**	250	10
3	k-means	0,73	0,74	0,68	0,77	0,74	0,71	0,76
3	HMM	0,75	0,75	0,75	0,78	0,77	0,83	0,76
3	SVM	0,72	0,73	0,74	0,78	0,8	0,78	0,77
3	Decision tree	0,73	0,68	0,67	0,75	0,77	0,76	0,73
3	Random forest	0,72	0,7	0,72	0,85	0,83	0,85	0,77
5	*Avg Accuracy*	0,73	0,72	0,712	0,786	0,782	0,786	0,758
5	k-means	0,77	0,79	0,81	0,75	0,75	0,75	0,85
5	HMM	0,83	0,85	0,79	0,9	0,88	0,92	0,74
5	SVM	0,74	0,74	0,87	0,77	0,75	0,75	0,88
5	Decision tree	0,66	0,7	0,64	0,79	0,79	0,77	0,74
5	Random forest	0,72	0,74	0,74	0,83	0,81	0,83	0,81
5	*Avg Accuracy*	0,744	0,764	0,77	0,808	0,796	0,804	0,804
7	k-means	0,62	0,56	0,59	0,7	0,7	0,7	0,67
7	HMM	0,7	0,72	0,72	0,78	0,81	0,81	0,91
7	SVM	0,67	0,64	0,62	0,67	0,67	0,67	0,75
7	Decision tree	0,62	0,54	0,67	0,7	0,67	0,67	0,59
7	Random forest	0,67	0,64	0,64	0,72	0,75	0,72	0,78
7	*Avg Accuracy*	0,656	0,62	0,648	0,714	0,72	0,714	0,74
10	k-means	0,7	0,66	0,74	0,77	0,77	0,81	0,7
10	HMM	0,74	0,7	0,7	0,88	0,88	0,88	0,81
10	SVM	0,66	0,66	0,7	0,88	0,88	0,92	0,74
10	Decision tree	0,62	0,66	0,66	0,85	0,85	0,81	0,77
10	Random forest	0,74	0,66	0,7	0,88	0,88	0,88	0,7
10	*Avg Accuracy*	0,692	0,668	0,7	0,852	0,852	0,86	0,744

Figure 9: Result for hardware parameter settings from accuracy for each model and lengths of time per observation. Data set 1 to 7.

set 9-11). Finally, one feature set excluding the mean amplitude was tested as the reliability for this feature can be discussed, and was excluded for the live prediction functionality considering only static and dynamic events (feature set 12). The best result for each row is put in bold text. The models were trained with normalised data and with 5% of the data removed with the assumption that the data set could contain outliers. Values were rounded to two decimal places but in reality many of the 1.00 values and 0.99 only differed in the third decimal place with less than 0.005. This small difference is probably not statistically assured.

As mentioned earlier the amplitude mean feature might not be reliable although it generates good results. It has been observed that the measured mean amplitude can vary a lot when the antennas move just a few centimetres. It is also suspected that an offset error exists that varies between each connection to a Pluto device. It has not been established if the unpredictable behaviour comes from the environment or the hardware. But if the error is random and introduced by the hardware a calibration could be made in the beginning of each session by removing the static mean value from all measurements during that connection.



DS8	DS9	D10	DS11	DS12	DS13	DS14	DS15
1 015 000 000	1 015 000 000	**915 000 000**	**915 000 000**	**915 000 000**	**915 000 000**	**915 000 000**	**915 000 000**
1 000 000	**1 000 000**	550 000	**550 000**	550 000	2 000 000	2 000 000	2 000 000
100	250	10	**100**	250	10	**100**	250
0,78	0,77	0,85	**0,83**	0,73	0,81	0,8	0,81
0,7	0,73	0,86	**0,87**	0,8	0,8	0,78	0,8
0,76	0,77	0,9	**0,9**	0,82	0,83	0,83	0,83
0,74	0,77	0,78	**0,83**	0,65	0,66	0,58	0,73
0,76	0,76	0,87	**0,85**	0,85	0,76	0,78	0,76
0,748	0,76	0,852	**0,856**	0,77	0,772	0,754	0,786
0,85	0,83	0,87	**0,83**	0,81	0,81	0,79	0,77
0,79	0,81	0,87	**0,87**	0,85	0,77	0,75	0,75
0,88	0,9	0,9	**0,9**	0,92	0,79	0,79	0,81
0,74	0,66	0,87	**0,83**	0,79	0,79	0,79	0,75
0,83	0,85	0,87	**0,85**	0,9	0,77	0,75	0,81
0,818	0,81	0,876	**0,856**	0,854	0,786	0,774	0,778
0,67	0,59	0,86	**0,83**	0,62	0,78	0,89	0,83
0,91	0,89	0,83	**0,83**	0,7	0,89	0,89	0,91
0,75	0,72	0,89	**0,89**	0,81	0,94	0,91	0,94
0,59	0,75	0,75	**0,64**	0,7	0,64	0,64	0,64
0,75	0,75	0,81	**0,72**	0,7	0,89	0,91	0,89
0,734	0,74	0,828	**0,782**	0,706	0,828	0,848	0,842
0,7	0,77	0,74	**0,74**	0,6	0,81	0,85	0,81
0,81	0,81	0,7	**0,7**	0,7	0,74	0,74	0,77
0,74	0,74	0,85	**0,85**	0,81	0,85	0,85	0,85
0,77	0,74	0,59	**0,59**	0,62	0,74	0,74	0,74
0,7	0,7	0,74	**0,77**	0,59	0,85	0,81	0,81
0,744	0,752	0,724	**0,73**	0,664	0,798	0,798	0,796

Figure 10: Result for hardware parameter settings from accuracy for each model and lengths of time per observation. Data set 8 to 15.

7.5 Machine Learning Models

From the results in Table 2 it can be seen that several sets of features perform well for the multi-hypothesis testing. Feature set 7 was one of the best performing and was selected to train the final models with. Training models with feature set 7 resulted in good models both for the multi- and binary hypothesis case. The accuracy for this set was not the highest, but as the difference was small to feature set 9 that performed best using only half as many features, it was favoured to reduce complexity of the final models.

Further, the result was quite similar with and without the calibration, even though the calibration affects the result. Because the models without the calibration performed better the final model was created without it. Some different percentages of outliers removed were tested 0%, 5%, 10%, 15% and 20%. Among these the average accuracy was best for 0% with the value 0.79, indicating that the test set does not contain many outliers. The top three models were

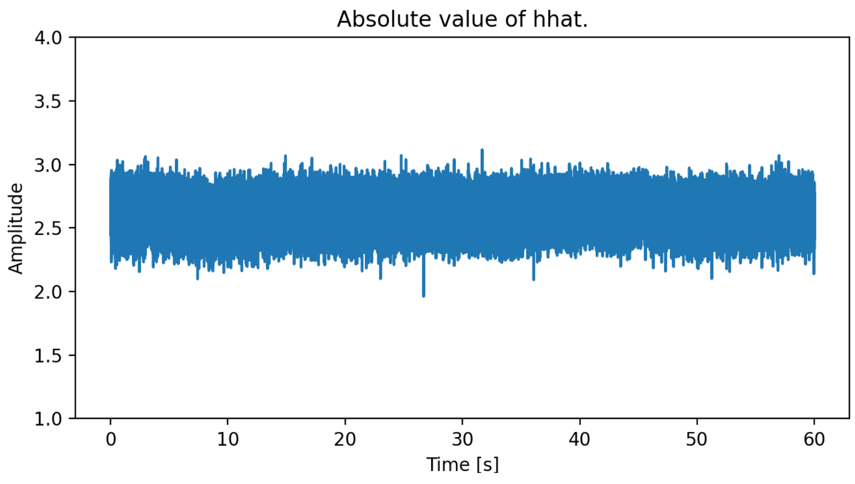


Figure 11: A plot of a raw CSI sample labelled as static.

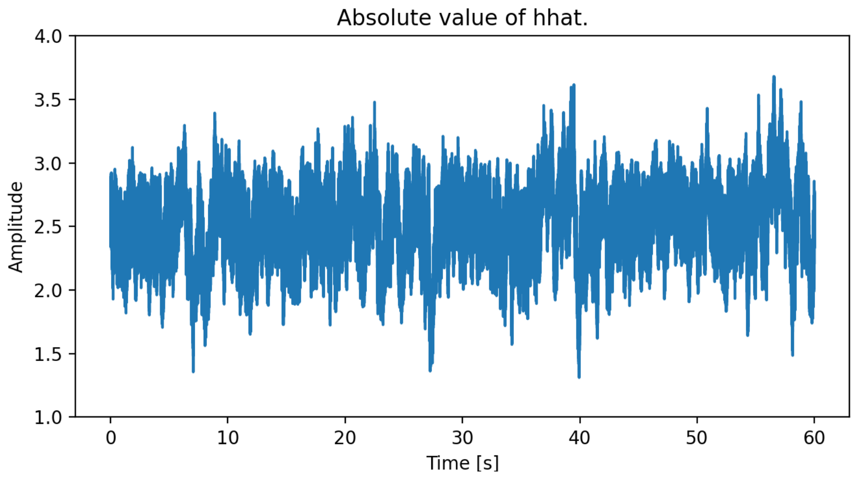
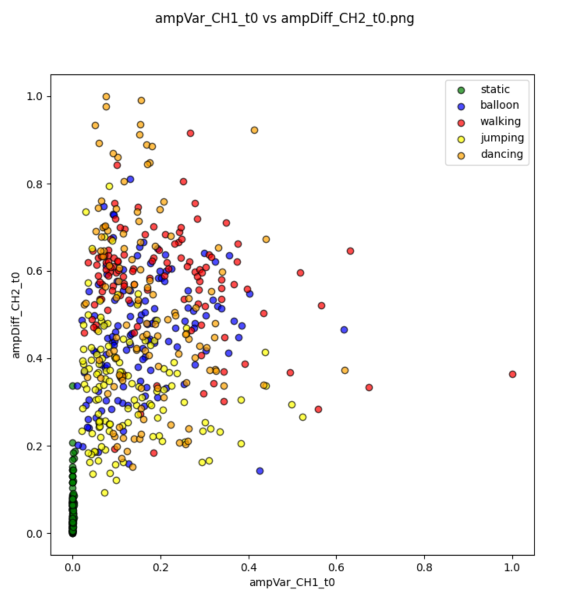
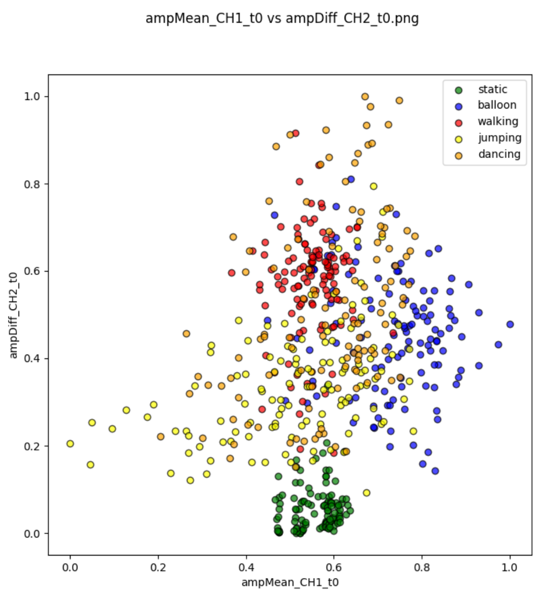


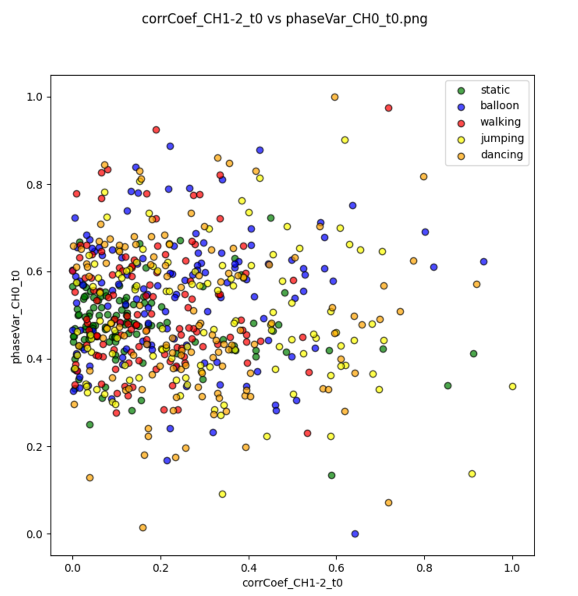
Figure 12: A plot of a raw CSI sample labelled as balloon.



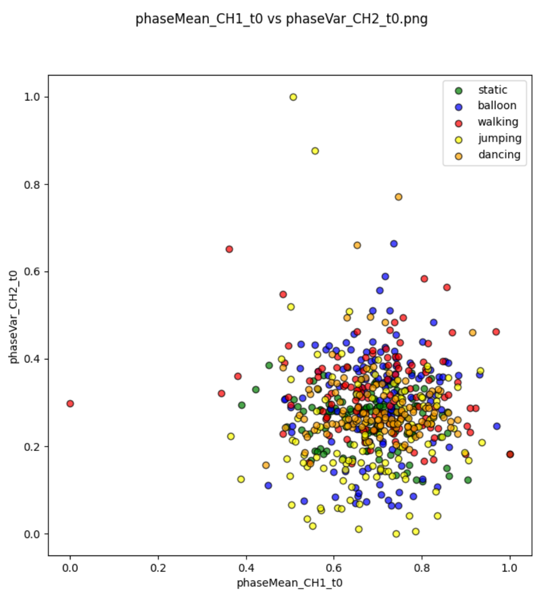
(a) Amplitude max difference plotted against amplitude variance.



(b) Amplitude max difference plotted against amplitude mean.



(c) Phase variance plotted against channel correlation coefficient.



(d) Phase variance plotted against phase mean.

Figure 13: Illustration of the distribution for all the feature values. Each feature is calculated for all channels but in these plots one channel per feature was selected.

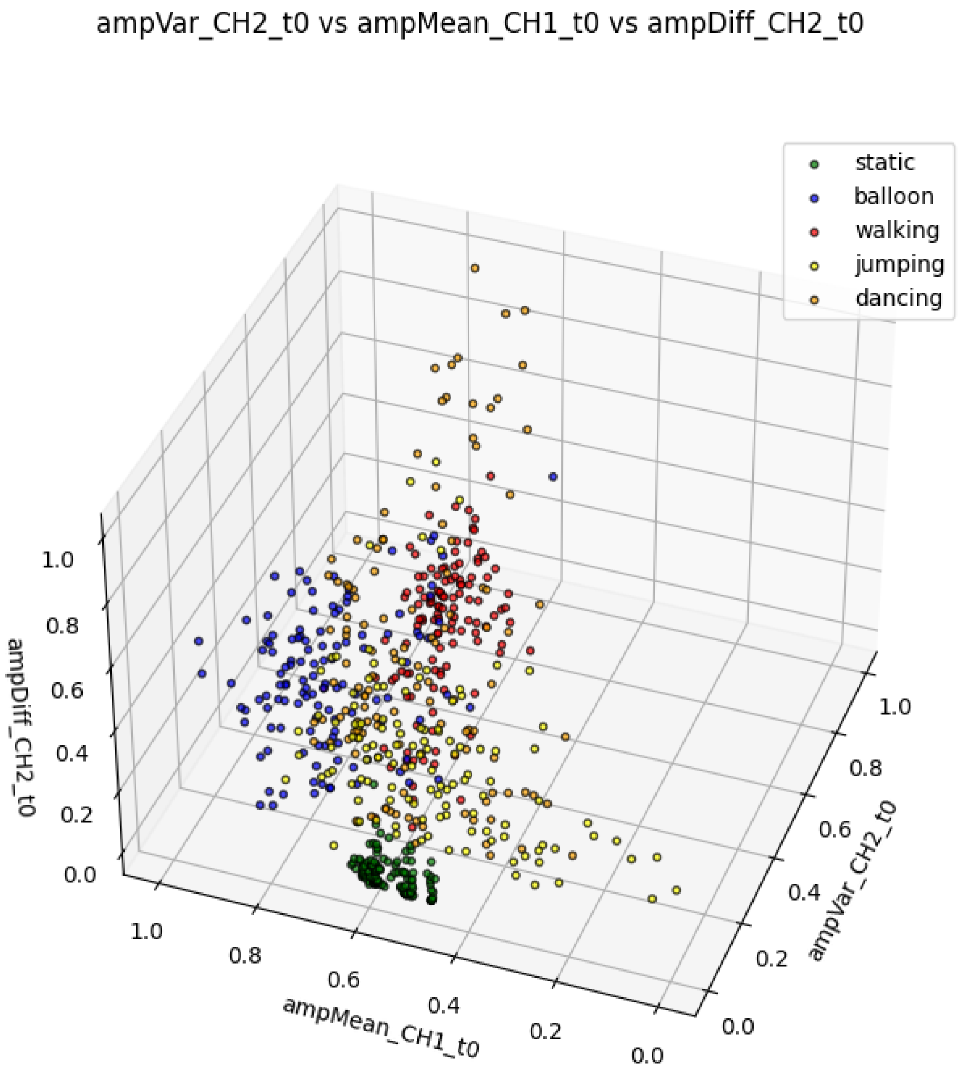


Figure 14: Distribution of feature values for the three features identified as best, one of three used channels shown. CHX in the variable name specifies for which of the channels the feature was calculated for.

Table 2: Summary of model performance for all models and different set of features.

Feature set	1	2	3	4	5	6	7	8	9	10	11	12
Mean amplitude	X						X	X	X	X	X	
Amplitude variance		X					X	X	X	X	X	X
Amplitude max difference			X				X	X	X	X	X	X
Channel correlation				X				X		X	X	
Mean phase					X			X	X		X	
Phase variance						X		X	X	X		
Avg. accuracy	0.71	0.56	0.61	0.38	0.50	0.42	0.79	0.79	**0.80**	0.77	0.76	0.63
Avg. F1-score static	0.94	0.99	**1.00**	0.70	0.92	0.55	**1.00**	**1.00**	0.99	0.99	**1.00**	**1.00**
Avg. accuracy (calibration)	0.66	-	-	-	-	-	0.74	**0.78**	0.76	0.74	0.76	-
Avg. F1-score static (calibration)	0.93	**-**	**-**	**-**	**-**	**-**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**-**

SVM with accuracy 84%, HMM with accuracy 83% and random forest with accuracy 83%. When training the model for only static and dynamic data all the models get 100% accuracy except k-means clustering that got an accuracy of 98%. See the results for test data in Figure 15. Figure 16 shows confusion matrices for the models trained.

To discriminate only between the static and dynamic classes many of the feature sets can be used as multiple have a high F1-score for the static class. A high F1-score indicates that almost all static observations where correctly classified and thaw few dynamic observations were incorrectly classified as static. For the binary models the features that showed best result for the F1-score would be most interesting to include, therefore feature set 12 will be used to train models for the binary classification. Removing the amplitude mean feature removes the possibility of this feature being unstable between sessions, which is good as the model also will be used to do classification on new samples collected. In Figure 17 the test data results can be shown for models trained with feature set 12 and all four dynamic classes labelled as dynamic.

To evaluate the models, it can also be interesting to look at fitted decision trees and which features that are most important during their formation. Figure 18 shows a tree for the binary hypothesis trained with feature set 12, Figure 28 in the appendix shows one tree trained with feature set 7.

For the pruning of the decision tree model minimal-cost complexity pruning was used. Figure 19 shows the accuracy plotted against the effective value of alpha, which is a measure of the complexity of the tree. To choose an appropriate model, the model with the highest test-score should be selected.

The random forest does not require so much manual tweaking but instead it can give insight about which features that are more important. In Figure 20 a plot over the mean decrease in impurity after splits with the features in feature set 8 can be seen. The black line shows the standard deviation for that feature's mean decrease in impurity for all individual trees in the random forest. From the plot it is suggested that the feature amplitude mean on channel one is the most important and generally the three feature types used for the final models (amplitude variance, amplitude mean and amplitude max difference) have high scores. Also, the phase feature on channel two seems to be good. For many features the standard deviation is quite high, but it can depend on the fact that many features are considered and that quite few samples are available. An important thing to have in mind is that this analysis is based on the performance

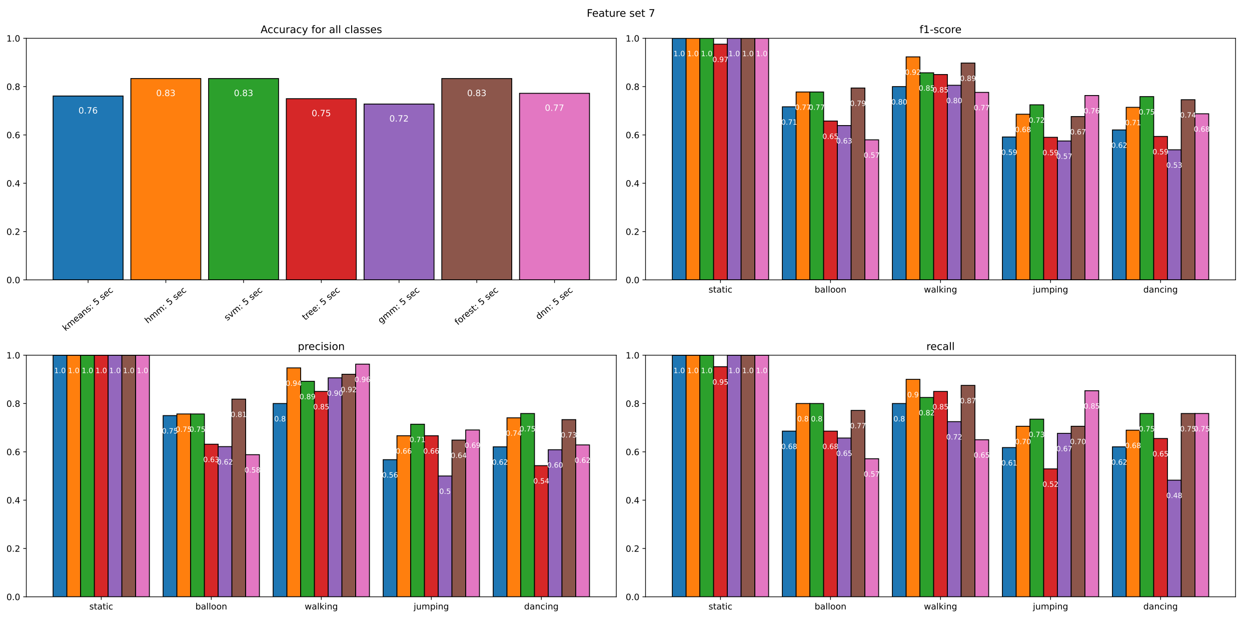


Figure 15: Model performance for feature set 7 when predicting on test data.

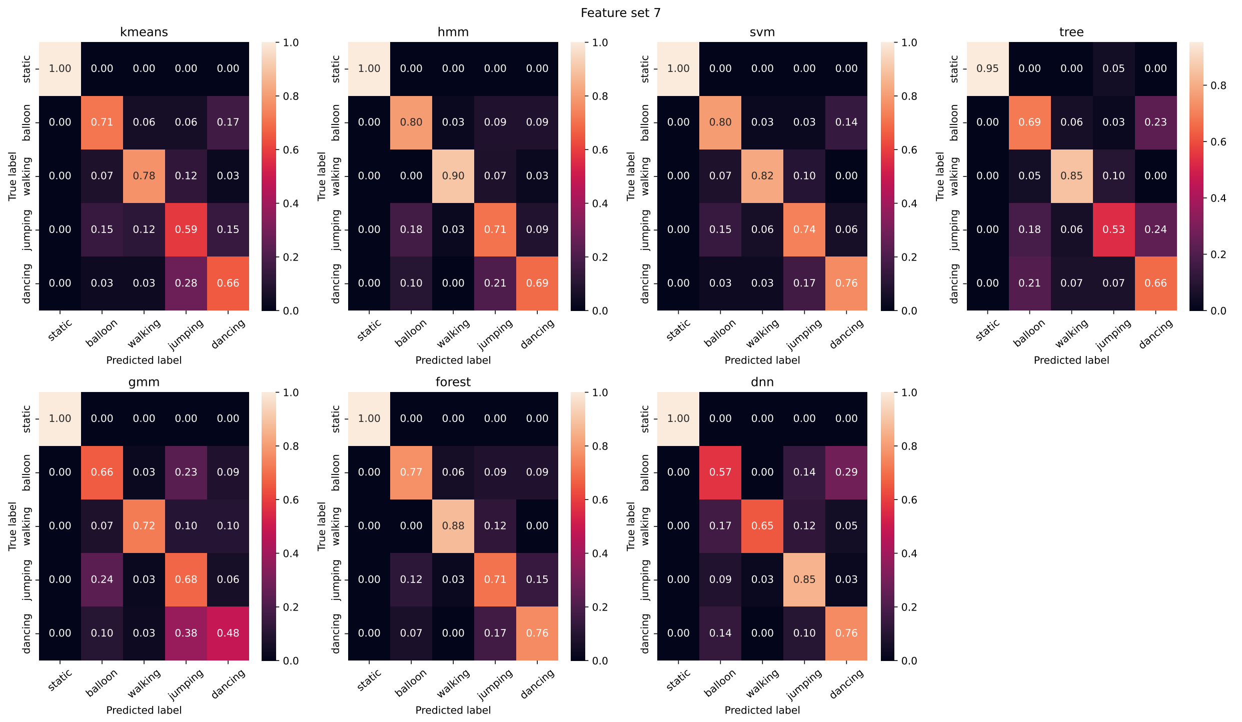


Figure 16: Confusion matrices for feature set 7 when predicting on test data.

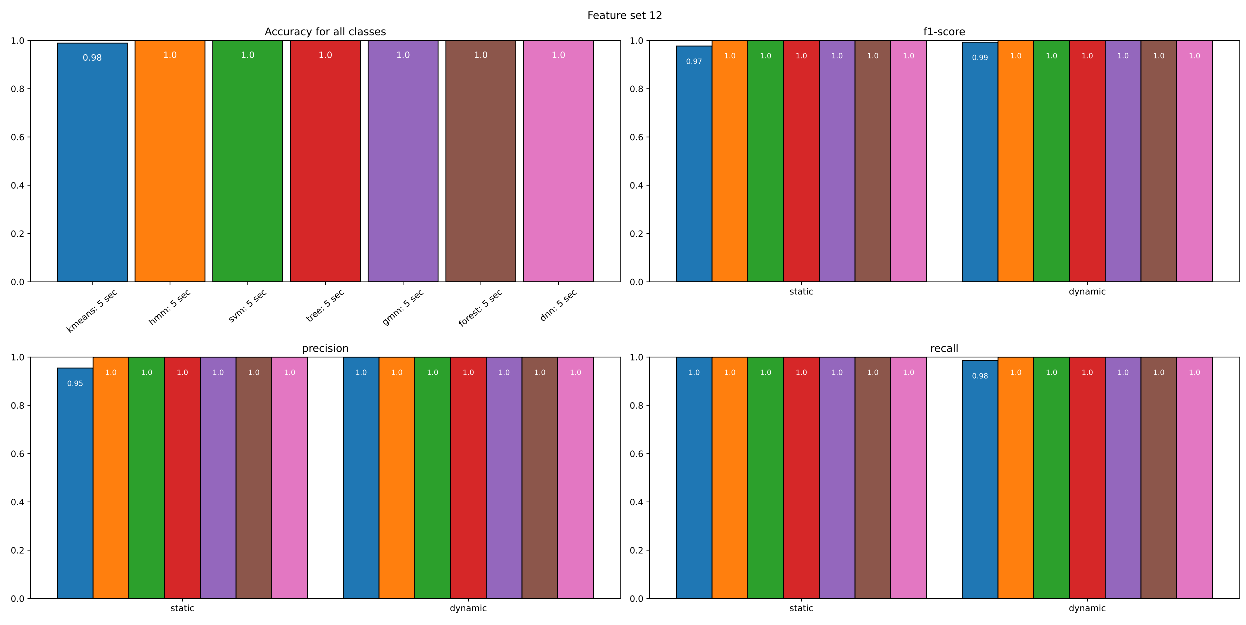


Figure 17: Model performance for feature set 12 when predicting on test data.

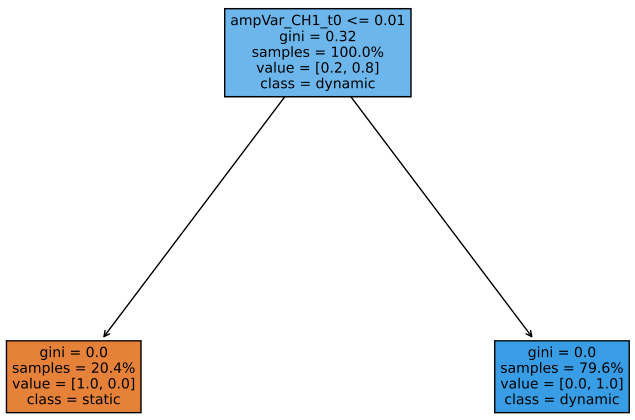


Figure 18: Decision tree model created using feature set 12 and post pruning. The classification is made with only one of the available six features.

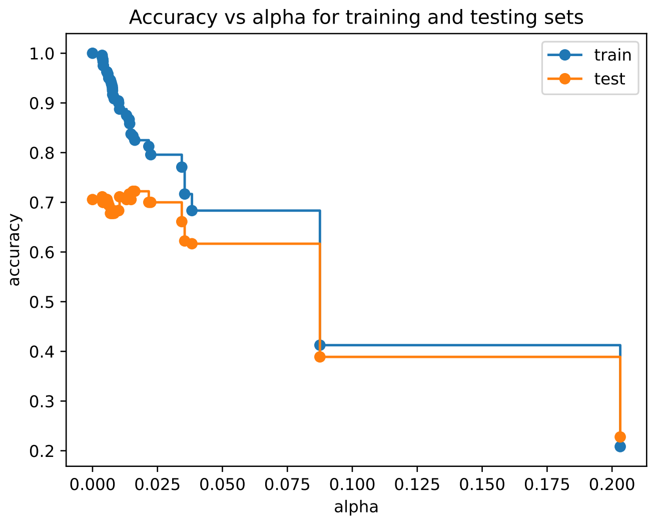


Figure 19: Plot over the cost-complexity pruning for the final decision tree model. The alpha that gives the highest accuracy value for the test data is used for the final model.

of the training data and might not be representative for the data set as a whole, but it can give a clue to which features that are important.

For the SVM model, three different kernels were used: a linear, a radial basis and a polynomial. Using grid search it was found that the radial basis kernel performed best, and the other hyper parameters were also tuned by a grid search to select the model that achieved the best accuracy for the training data. That the model performed best with a radial basis function was reasonable as the data was not linearly separable and tendencies to cluster exist.

7.5.1 *Learned Parameters of the Hidden Markov Model*

The HMM was one of the more complex shallow ML models used for classification, and it is therefore interesting to evaluate and interpret the parameters of the HMMs trained for classification.

A subset of the learned parameter for the HMMs representing the static and walking classes is shown in Figures 23 and 24 in Appendix A. The models for the other classes are also visualised in Appendix A. The nodes represent the hidden states, and the arrows with numbers the transition probabilities. The yellow numbers indicate initial probabilities, and the vector adjacent to the nodes are the emission means (three for each feature) amplitude variance, amplitude mean, amplitude max difference, phase mean, and phase variance. The results are discussed in Section 8.4.1.

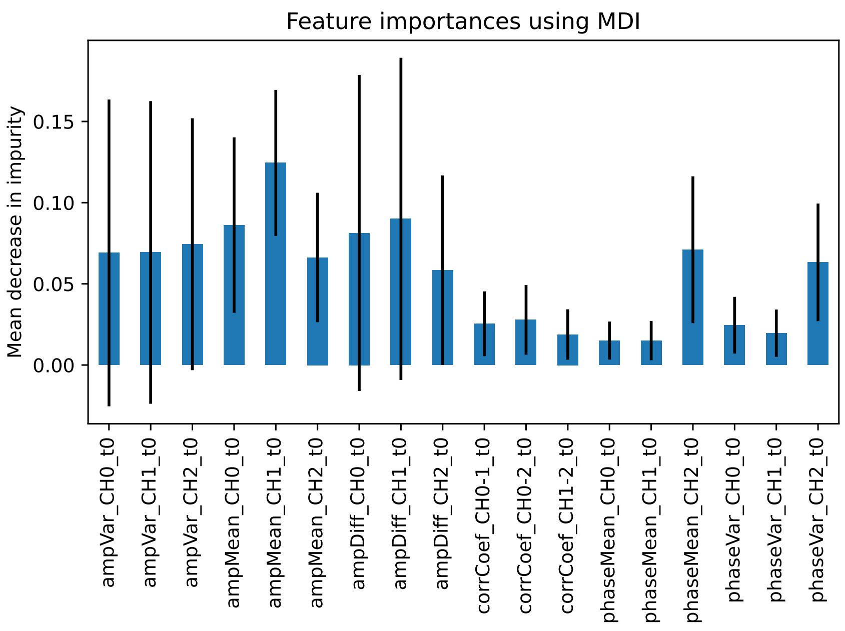
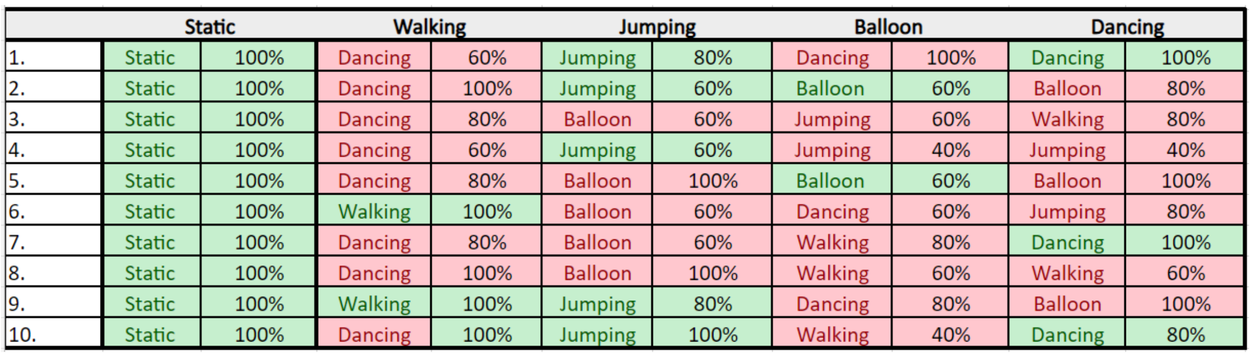


Figure 20: Mean decrease in impurity for the features used in the random forest model trained with feature set 8. The features used in the final models (feature set 7) are the ones with highest scores, also the phase on one channel seems to be useful.



	Static		Walking		Jumping		Balloon		Dancing	
1.	Static	100%	Dancing	60%	Jumping	80%	Dancing	100%	Dancing	100%
2.	Static	100%	Dancing	100%	Jumping	60%	Balloon	60%	Balloon	80%
3.	Static	100%	Dancing	80%	Balloon	60%	Jumping	60%	Walking	80%
4.	Static	100%	Dancing	60%	Jumping	60%	Jumping	40%	Jumping	40%
5.	Static	100%	Dancing	80%	Balloon	100%	Balloon	60%	Balloon	100%
6.	Static	100%	Walking	100%	Balloon	60%	Dancing	60%	Jumping	80%
7.	Static	100%	Dancing	80%	Balloon	60%	Walking	80%	Dancing	100%
8.	Static	100%	Dancing	100%	Balloon	100%	Walking	60%	Walking	60%
9.	Static	100%	Walking	100%	Jumping	80%	Dancing	80%	Balloon	100%
10.	Static	100%	Dancing	100%	Jumping	100%	Walking	40%	Dancing	80%

Figure 21: Result from live classification, for each class 10 classifications were done and the result for which class the model decides are presented in the table. The percentage stands for how the voting went e.g., 80% means that four out of five models gave the final result. Used models in this test was SVM, k-means, decision tree, random forest and GMM.

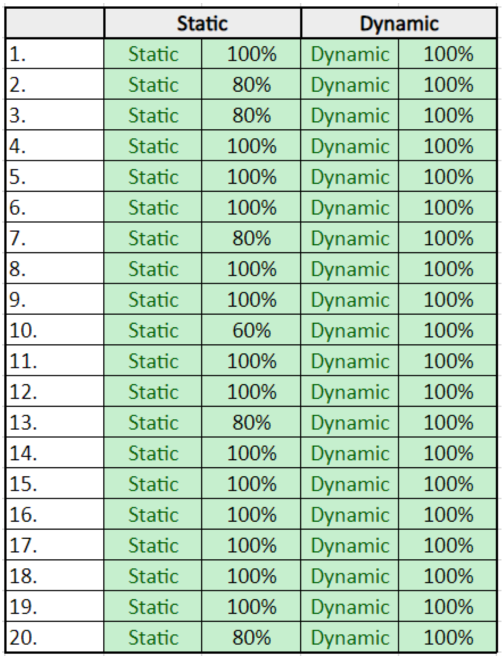
7.6 Live Classification

All models were trained with the collected data set using feature set 12 (presented in Table 2) to obtain the result for live classification. Two tests were performed, one with all classes and one where the classes walking, jumping, balloon and dancing were merged with the setup into one class called dynamic. Several live classifications were done in the setup room and the results are presented below. The test result for all classes can be seen in Figure 21 and the result for static and dynamic can be seen in Figure 22. The live predictions get a very good result when only classifying static and dynamic. When all classes are used the model is having a hard time to figure out what type of dynamic class it is beyond the static class. A theory why the classifications gets worse in live classification than in the evaluation of the models is provided in the discussion in Section 8.

8 DISCUSSION

8.1 Hardware Parameters

There are some uncertainties in the investigation of hardware parameters. To test the different hardware parameters, it was necessary to collect new data for every new parameter configuration. Since there were five different classes and 15 different configurations of hardware parameters it was time consuming to collect all data. It was decided to collect 3 minutes of data for each class, which might have been too little data to get reliable results. When doing the 10 second classification this becomes only 18 observations per class. The time aspect was also the reason for why more configurations were not tested. A small data set for each configuration adds more uncertainty, it can be hard to conclude if the differences depend on the different hardware parameter settings or the differences in data collection. Data set 10 and 11 get quite similar results but data set 11 is the chosen one since the binsize is larger which leads to less data to store. This is the reason why this test is still used as a basis. It might not be true to say with certainty



	Static		Dynamic	
1.	Static	100%	Dynamic	100%
2.	Static	80%	Dynamic	100%
3.	Static	80%	Dynamic	100%
4.	Static	100%	Dynamic	100%
5.	Static	100%	Dynamic	100%
6.	Static	100%	Dynamic	100%
7.	Static	80%	Dynamic	100%
8.	Static	100%	Dynamic	100%
9.	Static	100%	Dynamic	100%
10.	Static	60%	Dynamic	100%
11.	Static	100%	Dynamic	100%
12.	Static	100%	Dynamic	100%
13.	Static	80%	Dynamic	100%
14.	Static	100%	Dynamic	100%
15.	Static	100%	Dynamic	100%
16.	Static	100%	Dynamic	100%
17.	Static	100%	Dynamic	100%
18.	Static	100%	Dynamic	100%
19.	Static	100%	Dynamic	100%
20.	Static	80%	Dynamic	100%

Figure 22: Result from live classification, for each class 20 classifications were done and the result for which class the model decides are presented in the table. The percentage stands for how the voting went e.g. 80% means that four out of five models gave the final result. Used models in this test was SVM, k-means, decision tree, random forest and GMM.

that this hardware setting is the best, but what can be said is that it is not worse than the rest. A lower sample rate and bigger binsize makes the data smaller, which motivates the choice.

8.2 Definition of Activities

Classification of activities only makes sense when the activities are well-defined in terms of movement. It is not unreasonable to view dancing as a way of jumping around, and therefore the activities in this study have been limited to the description in Section 3.1. To pose a real challenge to the system the activities must have some form of similarity. In Figure 13b, jumping and dancing are that overlap the most. The activities were performed by different persons from the project group, and the samples will probably be more varying than if a single person would have performed the movements.

8.3 Features

Six different features were used to train the models. It could have been interesting to test additional features, but with the data available the ones implemented was the ones found most interesting to test. Also, it could have been interesting to use data from multiple sub-carriers if the implementation of OFDM signals had reached the final product.

Using the amplitude of the estimated CSI was the first thing tested and by using only this feature good results were achieved. The amplitude variance gave a clear division between static and dynamic events, this was not surprising as the CSI varied a lot more when there was movement in the room. Also the amplitude max difference gave a clear division between static and dynamic events, which also can be explained by the fact that the estimated CSI was almost constant for static environments but varied a lot for the dynamic environments.

Regarding the amplitude mean it was uncertain if the feature gave a good result only from the event or if it also could be explained by sensitivity to environmental changes or random offsets introduced by the hardware. Using the calibration method presented in Section 5.1 generally resulted in a worse result and therefore it was not used in the final models. Not making the calibration can have resulted in that the measurements were spread out more in the feature space. Very different amplitude offsets between sessions would result in clear clusters for each session. Since all observations during one session was of the same type it follows then that all observations in such a cluster belongs to the same class. Since some ML models will have a much easier time to classify observations correctly if all classes lie in clear clusters a misleading good result might be obtained. From the results and other tests performed during development it can be stated that this feature is probably not useful if the detector would be used in new environments without collecting new data for training.

Apart from the simple features using the amplitude or phase for one channel at a time, a feature for the correlation coefficient between pairs of channels was tested. The feature is based on the correlation between the amplitude values of the CSI data and is calculated for each pair of channels. This feature did not perform well on its own and for the multi hypothesis testing the feature was excluded as it did not improve the result.

8.4 Machine Learning Models

When evaluating the results for the trained models evaluated at test data, the result for every model and class gets very good which can be seen in Figure 15. The result shows that the models can differentiate between classes that can be quite similar e.g., dancing and jumping. But when looking at the live results in Figure 21 it is possible to see that the result is not as good. One reason might be that in the data collection there are some irregularities that differentiate between each data collection. One variable that seems to (randomly) change between data collections is the amplitude. That each data collection have its own (what it seems like) random starting point in amplitude could be an explanation that the different classes can be separated in a good way when trained and tested on data collected on the same time. When doing the classification on live collected data the amplitude is different and it is not possible to differentiate on the different dynamic classes. One theory is that the hardware is the cause for this error rather than the models.

When looking at the result for the trained models and ignoring the fact that the live classification gets bad, all the models give a very good result. There is no clear model that is better than the others, but one could argue that since there is no clear distinction in the results for the different models the simpler models such as SVM and decision tree are better since they can get the same good result as the more complex ones as HMM and DNN. Among the simpler models the decision tree might be too simple as it does not even take all features into account, for example it used only one channel's maximal amplitude difference to classify all static events, see Figure 28. For the binary case only one out of six available features were used, see Figure 18. In contrast SVM uses all features and in a random forest all available features will most probably be used as multiple trees are created. The models are trained on quite few features which also support the theory on using more simple models. One model that could be interesting to investigate more is the HMM which is the only model that takes the time aspect into consideration. Since the detector is supposed to classify movement, it is quite intuitive to think that a model that also take time into account would perform better than those that do not. Another interesting to note is that the GMM performed worse than the k-means clustering, which was also the case when running the algorithms on average. Increasing the number of training restarts might make the results more even.

8.4.1 *Evaluation of Hidden Markov Model Parameters*

As stated earlier, the reason for investigating the parameters of the HMMs is that it is a more complex model. To get a very good result the emissions must be sufficiently distinct between the classes of activities, early in the experiments with amplitude mean and variance as features it was found that there was significant overlap between the dynamic classes.

In the model representing static, one can see that the amplitude variance has zero mean for all states, and that one hidden state is much more likely than the others. The dominant states Z^1 and Z^2 have very similar emission means. Since there is not too much activity in an ideal static environment, it is not too surprising that the static model gives similar hidden states. The model for walking notably has more varied estimates of the amplitude variance, and the model describes a higher likelihood to transition between hidden states compared to the model for static. The covariances between features were estimated to be close to zero.

Each model presented here was trained on about 70 samples containing 15 features, and the number of parameters to be estimated with three hidden states is about 15 (9 transition probabilities, 3 initial probabilities, and 3 mean values and 3 variance values per feature, not counting covariances between features). The ratio of data against parameters is quite small, which might impact the result. As discussed earlier, the impact of outliers might affect the learned parameters, as the model might use a single data point to estimate a hidden state.

Adding more hidden states would probably not yield better results for the models that have states with similar emission distributions, like the model representing static. More hidden states would also imply learning more parameters, which can be a drawback, especially if data is limited. Since the less complex ML models perform well on the collected data set, it is not certain that more complexity is the right answer unless there is a change in the feature set. It would be interesting to work with proper phase data, and sub-carrier correlation over time to make predictions.

Looking closer at the parameters of jumping and dancing (Figures 26 and 27), one may get the impression that the learned models reflect the activities they represent. The model for jumping has low rates of transition between states, while the dancing model has somewhat higher values. In theory, the hidden states are related to some unknown mode of the environment, each of which specifies a distribution over the observed values. The activity jumping is itself less varied than dancing, as the received signal might depend on the position the jumping person (which does not change too much in contrast to dancing). The emission means of these models are harder to interpret.

8.5 Live Prediction

The result for the live classifications showed that the system can do live classifications with a very good accuracy when just discriminating between a static and a dynamic environment. This follows what could be expected given that the F1-score for the static class was 1.00 for the selected feature set and that almost all models trained with these features had an accuracy of 1.00 in this binary hypothesis testing, see Figure 17. Why the result for the live multiple hypothesis case is worse than the result when using the original test set can be due to many reasons. One source of error is that two models were excluded in the live case. Another is that the live observations were made with only two individuals performing the activities and they might not be representative for the overall distribution. The amplitude mean feature was excluded during the live testing because the result was much better when it was. Why that is the case is unknown but as discussed earlier, it can be suspected that this feature is more sensitive to environmental changes than the other, or the hardware introduce random errors affecting this feature more than the others.

8.6 DNN Model

The results of the DNN model varied a lot depending on how it was calibrated. It was very sensitive to changes in the number of hidden layers and nodes. There is no perfect formula to follow to calibrate all the parameters of DL models, they must be found by testing and evaluating. When a suitable number of hidden layers and nodes was found, the DL model achieved good results.

To really get the most out of DL, much more data should be used, but with limitations in the data collection, that was not possible. Better results might be achieved if more data were to be used. The DL model was able to perform as good as many of the other models, but because the DL models are more complex, it was not worthwhile using them in a project of this size.

8.7 Experimental Errors and Difficulties

When interpreting results, it is important to think about possible sources of errors. When the data collection was performed, it was noticed the amplitude for the received signal shifted greatly between data collection runs. This for example, made it difficult to interpret the result from our ML models correctly. Was it possible to detect a different activity because a pattern was found? Or was it because of the amplitude shifting, and why was it not the same or roughly the same each run?

By evaluating the first training environment, which was the lab room, possible sources of errors could be:

- the quality of the antennas,
- the amount of electronics that were present in the room,
- the amount of people in the room and outside the premises,
- the temperature of the devices,
- the length of the antenna cables,
- the processing of the computers, and
- the placement of devices.

It was noticed that moving a device just a few centimetres, the results would change. Thus, small changes in the (static) environment led to huge differences in the signal propagation.

Taking these factors into consideration, along with the unpredictable amplitude changes, it proved difficult to identify errors in our controlled environment. It was not easy to perform changes to just a single aspect, and then interpret the outcome. To the best of the group's abilities, it was attempted to identify any errors in the lab room.

A decision was then made to swap to another room, an office room and collect data there. By that time, the number of features had increased which led to different results, and the amplitude changing was still present in the new environment. This led us to believe that the errors might not be from the data sets, but rather external.

8.8 Graphical User Interface

To integrate each part and to allow a user to easily control the system, a GUI was produced. The idea was to have a simple GUI where the user could start a data collection, create a model, see results, and start a live classification. To that end, the produced GUI delivers on those points.

The chosen package for the GUI in Python was Tkinter. Tkinter is a basic package that provides all the necessary building blocks for a proper GUI. There were other candidates, for example PyGame and PyQT, but ultimately Tkinter was chosen since it was the traditionally used package.

The idea for the GUI was that it should be as close to a marketable product as possible, with as little hard coding as possible, error handling, and proper threading/multiprocessing. The GUI can identify how many data sets a user can choose from and give a warning if none are present. Consideration about what could go wrong and to do proper error handling has been somewhat implemented, but not fully.

If the structure of the GUI had been better from the start, proper threading would have been easier to implement, but plenty of knowledge about front-end and back-end development has been gained throughout the project.

When the decision to swap to SIMO from SISO was made, one notable challenge was discovered. To create and run multiple Pluto devices using the multiprocessing package, it had to be run as the main script. This meant it could not be called as a function but had to be run as either a separate script or by using subprocessing. The next challenge then was to funnel the output from those functions through a PIPE which proved troublesome and time consuming at the time to implement. The workaround used was to call the script by using an OS-command.

8.9 Channel Evaluation

To evaluate the channel over which the signal was sent, the system computes the BER of the received signal. The same technique for sending a signal (using an OFDM signal and offset compensation at the receiving end) was not used when collecting data. Although the BER was low, it is not possible to say for certain how good. However, one may argue that since the obtained data (features) reflected a difference between static and dynamic environments, and since the ML models produce a high accuracy on classification, the channel estimates are useful.

8.10 Technical Challenges

At first the group worked with a given script in MATLAB that implemented time and carrier frequency offset compensation, and later translated the functions into Python. One challenge that was encountered was that an unknown error corrupted the sent signal so that the compensation algorithms were thrown off and the signal could not be recovered. Switching to another Pluto SDR device could sometimes resolve this issue for a while, until it failed again. This error only manifested when using Python, and was never encountered when running the MATLAB script. It would be possible to handle the hardware and data collection in MATLAB, and use Python libraries for ML. Using Python, the data type of the received signal is complex integer, which makes the received samples less exact if the amplitude is small. In MATLAB the values are floating point between zero and one.

During data collection, the received signal amplitude could differ from the previous collected signal by a significant amount even though there was no substantial change in the environment or hardware as far as the group could tell. This meant that using the mean amplitude as a feature was hard, especially in the case of live prediction.

8.11 Future Work

There are a few potential improvements that could have been implemented to get a better performance of the system. As mentioned above, using several sub-carriers by sending OFDM signals (instead of single carrier signals on three channels) would give the opportunity of using for example the correlations between sub-carriers as features in the ML part. To make the system more robust, more data would have been needed, and several more antenna setups and different locations would preferably have been investigated to ensure that the used approaches work in different environments. This could also have given more insights into in which situations the implemented system is most useful. One thing that would not have improved the performance, but could have made the project work substantially easier, would be to automatically save the collected data to a remote database or similar. This would have removed the inconveniences of uploading, downloading, reading and writing from/to large files.

9 CONCLUSIONS

It can be concluded that the methods used in this project are good at classifying very specific and well-defined classes such as static and dynamic. It is harder for the models to differentiate between multiple dynamic classes. The reason for this could be that the different dynamic classes are more or less clearly defined, but they are also designed to be similar in order to challenge the system. Another explanation could be that the collection of data differs between each data collection.

As can be seen in the plots in Section 7.4 it is possible to see that the classes are not linearly separable, the data points might not be separable in any feature space, and that makes it hard to achieve a higher accuracy.

The detector that does live classifications on the classes static and dynamic can do so with very high certainty. The models are probably overly complex but provides an accurate result. The features that are most important when classifying static and dynamic environment is amplitude max difference and amplitude variance.

The Pluto SDR devices have been both easy and hard to work with. Easy to plug in and start collecting data, but hard to handle when needing to do more signal processing, for example the BPSK. For the signal processing MATLAB would be a better choice of programming language than Python. But when building ML models Python is a good choice.

A challenge in this project have been to get the hardware to work, collecting data and train models in parallel. All parts have been very dependent on the hardware which have slowed down the process. Important key take aways from this project is the importance on high quality data and that one must have knowledge about the data collection and the data to be able to perform good ML modelling.

A APPENDIX

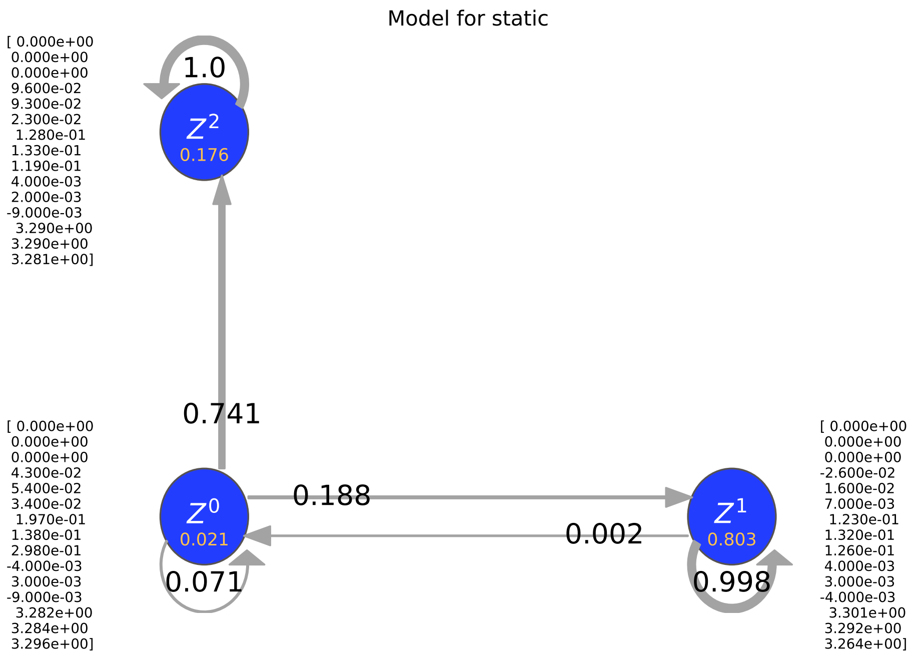


Figure 23: Markov chain over hidden states for the model representing static, with initial probabilities in yellow, and emission means of each state adjacent to the node. Arrows represent the transition probabilities. The emission mean values (from 3 channels for each feature) represent the features amplitude variance, amplitude mean, amplitude max difference, phase mean, and phase variance.

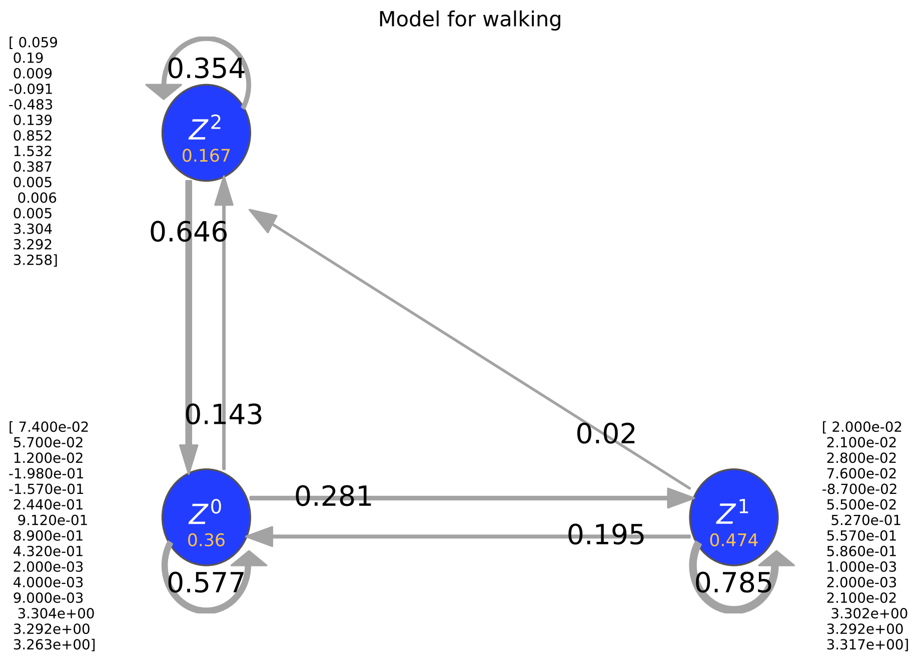


Figure 24: Markov chain over hidden states for the model representing walking, with initial probabilities in yellow, and emission means of each state adjacent to the node. Arrows represent the transition probabilities. The emission mean values (from 3 channels for each feature) represent the features amplitude variance, amplitude mean, amplitude max difference, phase mean, and phase variance.

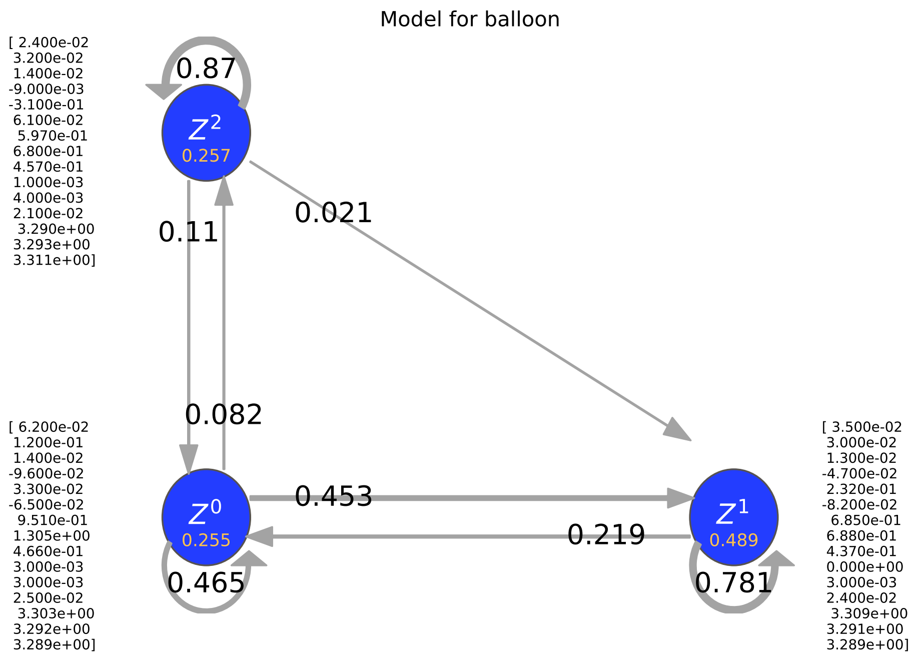


Figure 25: Markov chain over hidden states for the model representing balloon, with initial probabilities in yellow, and emission means of each state adjacent to the node. Arrows represent the transition probabilities. The emission mean values (from 3 channels for each feature) represent the features amplitude variance, amplitude mean, amplitude max difference, phase mean, and phase variance.

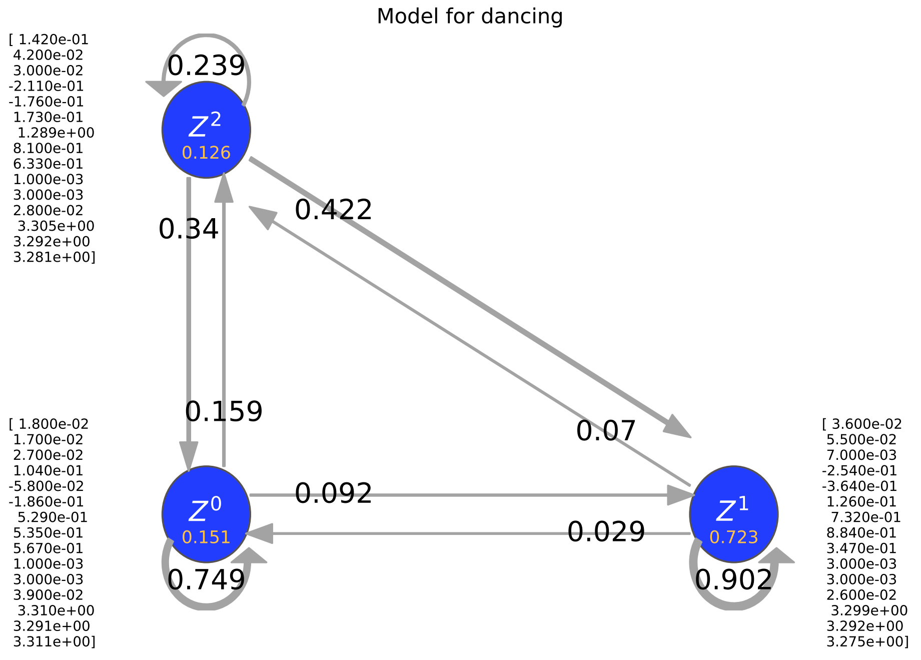


Figure 26: Markov chain over hidden states for the model representing dancing, with initial probabilities in yellow, and emission means of each state adjacent to the node. Arrows represent the transition probabilities. The emission mean values (from 3 channels for each feature) represent the features amplitude variance, amplitude mean, amplitude max difference, phase mean, and phase variance.

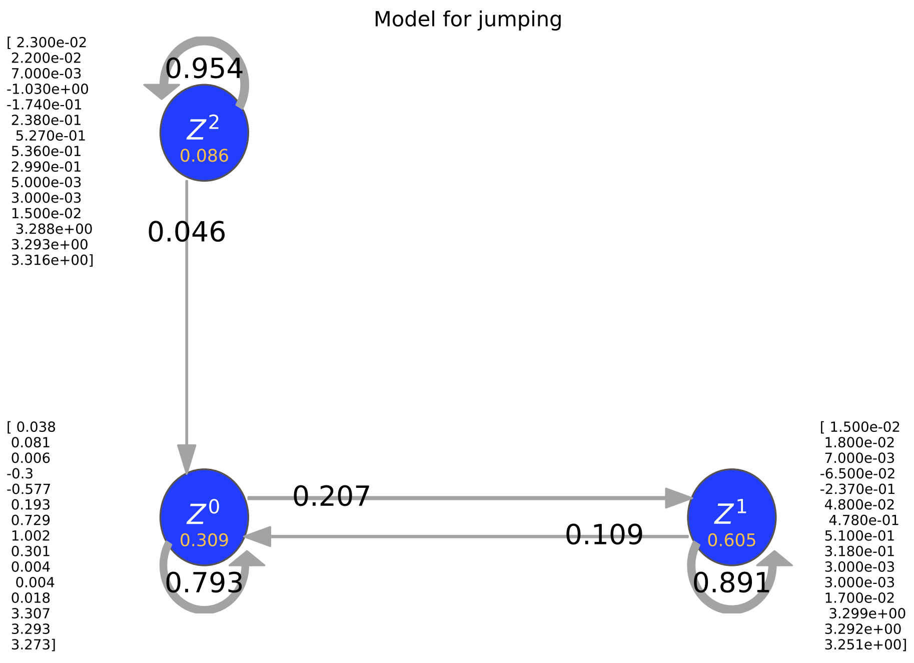


Figure 27: Markov chain over hidden states for the model representing jumping, with initial probabilities in yellow, and emission means of each state adjacent to the node. Arrows represent the transition probabilities. The emission mean values (from 3 channels for each feature) represent the features amplitude variance, amplitude mean, amplitude max difference, phase mean, and phase variance.

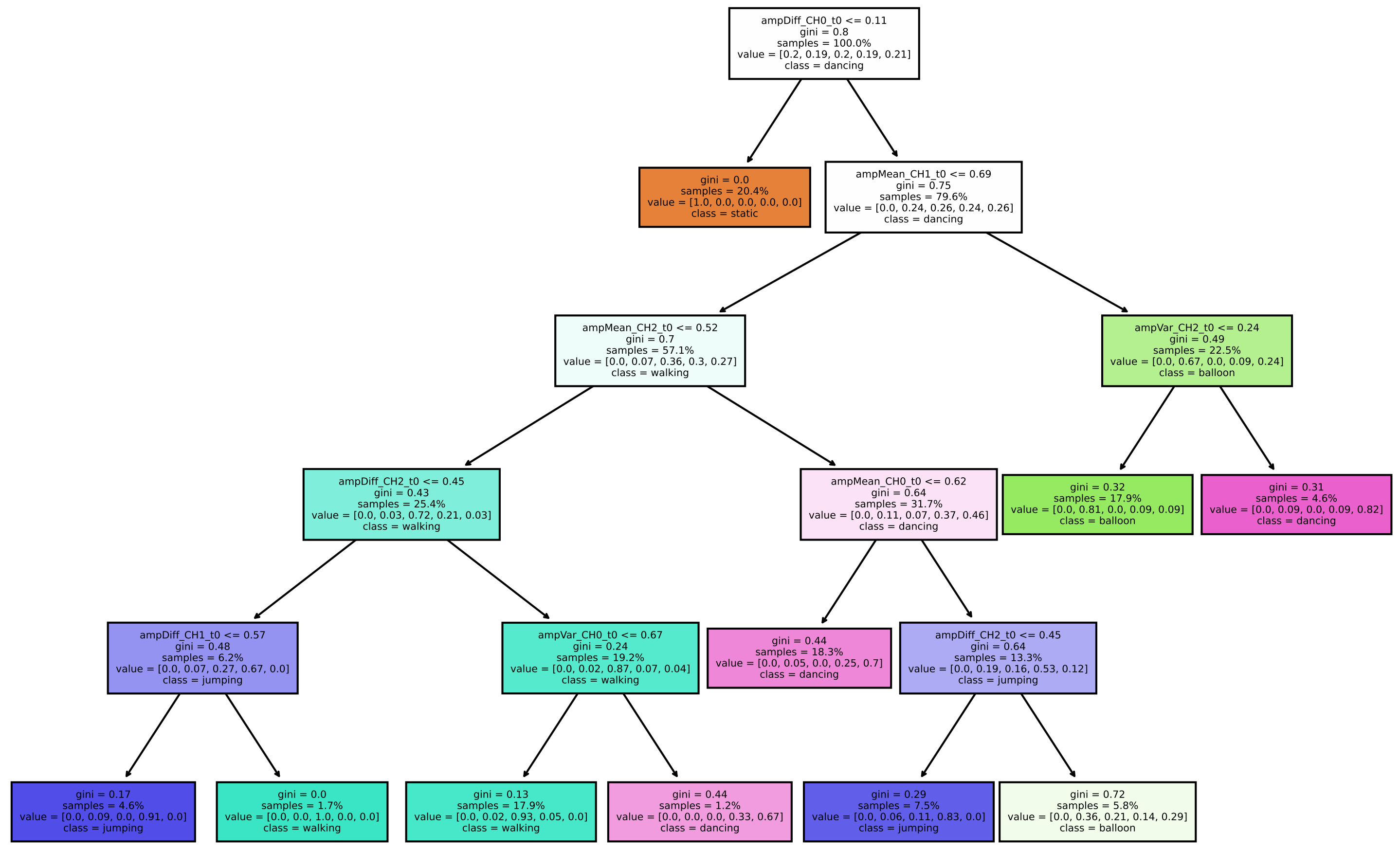


Figure 28: Decision tree model created using feature set 7 and post pruning. It can be noted that classification of the static class is made based on only one feature.

REFERENCES

[1] I. Nirmal, A. Khamis, M. Hassan, W. Hu, and X. Zhu, "Deep Learning for Radio-Based Human Sensing: Recent Advances and Future Directions," *IEEE Communications Surveys Tutorials*, vol. 23, no. 2, pp. 995–1019, 2021.

[2] S. Yousefi, H. Narui, S. Dayal, S. Ermon, and S. Valaee, "A Survey on Behavior Recognition Using WiFi Channel State Information," *IEEE Communications Magazine*, vol. 55, pp. 98–104, 2017.

[3] S. Liu, Y. Zhao, and B. Chen, "WiCount: A Deep Learning Approach for Crowd Counting Using WiFi Signals," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, 2017, pp. 967–974.

[4] Ding, Jianyang and Wang, Yong, "WiFi CSI-Based Human Activity Recognition Using Deep Recurrent Neural Network," *IEEE Access*, vol. 7, pp. 174 257–174 269, 2019.

[5] X. Chen, C. Ma, M. Allegue, and X. Liu, "Taming the inconsistency of Wi-Fi fingerprints for device-free passive indoor localization," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[6] Y. Tian, G.-H. Lee, H. He, C.-Y. Hsu, and D. Katabi, "RF-Based Fall Monitoring Using Convolutional Neural Networks," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 2, no. 3, Sep. 2018.

[7] Y. Ma, G. Zhou, and S. Wang, "WiFi Sensing with Channel State Information: A Survey," *ACM Comput. Surv.*, vol. 52, no. 3, Jun. 2019.

[8] Z. Wang, K. Jiang, Y. Hou, W. Dou, C. Zhang, Z. Huang, and Y. Guo, "A Survey on Human Behavior Recognition Using Channel State Information," *IEEE Access*, vol. 7, pp. 155 986–156 024, 2019.

[9] W. Wang, A. X. Liu, M. Shahzad, K. Ling, and S. Lu, "Understanding and Modeling of WiFi Signal Based Human Activity Recognition," ser. MobiCom '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 65–76.

[10] S. Sen, B. Radunovic, R. R. Choudhury, and T. Minka, "You Are Facing the Mona Lisa: Spot Localization Using PHY Layer Information," ser. MobiSys '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 183–196.

[11] L. Chen, J. Xiong, X. Chen, S. I. Lee, K. Chen, D. Han, D. Fang, Z. Tang, and Z. Wang, "WideSee: Towards Wide-Area Contactless Wireless Sensing," in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, ser. SenSys '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 258–270.

[12] W. Li, B. Tan, Y. Xu, and R. J. Piechocki, "Log-Likelihood Clustering-Enabled Passive RF Sensing for Residential Activity Recognition," *IEEE Sensors Journal*, vol. 18, no. 13, pp. 5413–5421, 2018.

[13] M. Andersson, E. Beskow, E. Grundin, R. Mannberg, J. Nilsson, G. Suihko, and J. Qu, "Detection of static and dynamic indoor environments: Requirement specification, 2021."

[14] T. Ulversoy, "Software defined radio: Challenges and opportunities," *IEEE Communications Surveys Tutorials*, vol. 12, no. 4, pp. 531–550, 2010.

[15] *ADALM-PLUTO for End Users.* Wilmington, MA, USA: Analog Devices, 2021, Accessed: Sep. 17, 2021. [Online]. Available: https://wiki.analog.com/university/tools/pluto/users

[16] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. Springer New York Inc., 2001.

[17] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[18] S. Marsland, *Machine Learning - An Algorithmic Perspective.*, ser. Chapman and Hall / CRC machine learning and pattern recognition series. CRC Press, 2009.

[19] *hmmlearn*. Accessed: Dec. 19, 2021. [Online]. Available: https://hmmlearn.readthedocs.io/en/latest/

[20] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[21] M. Andersson, E. Beskow, E. Grundin, R. Mannberg, J. Nilsson, G. Suihko, and J. Qu, “User manual, 2021.”