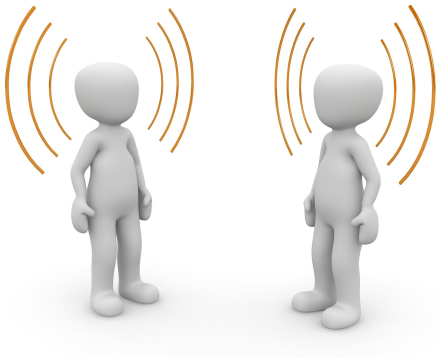


# Technical Documentation

Henrik Ahlinder, Sebastian Andersson, Martin Dahl, Christian Gustavsson,
Joel Henneberg, Tiger Kylesten.

December 16, 2022

Version 1.0



Status

| | | |
|---|---|---|
| Reviewed | Martin Dahl | 2022-12-16 |
| Approved | Danyo Danev | |



## Project Identity


Group E-mail: CDIOProjektgrupp@liuonline.onmicrosoft.com

Homepage: wavecounter.xyz

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: Sai Subramanyam Thoota
Phone: +46 (0)13-28 68 59
E-mail: sai.subramanyam.thoota@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** |
|---|---|---|
| Christian Gustavsson | Project Manager, PM | chrgu102@student.liu.se |
| Henrik Ahlinder | Interface Manager, IM, and deputy CD | henah490@student.liu.se |
| Joel Henneberg | Chief of Hardware, CH, and deputy PM | joehe862@student.liu.se |
| Martin Dahl | Documentation Manager, DM | marda545@student.liu.se |
| Sebastian Andersson | Test Manager, TM | seban218@student.liu.se |
| Tiger Kylesten | Chief of Design, CD | tigmo832@student.liu.se |



# CONTENTS

## DOCUMENT HISTORY

| Version | Date | Changes made | Sign | Reviewer |
|---|---|---|---|---|
| 0.1 | 2022-11-18 | Document skeleton | PM | - |
| 0.5 | 2022-12-12 | First draft | PM | Project group |
| 1.0 | 2022-12-16 | Approved by project group | PM | DM |



# 1 INTRODUCTION

In recent years a lot of attention has been given to detecting human activity with WiFi-signals. This has been a topic that not much progress had been made with traditional machine learning methods but with the rise of deep learning methods, the interest in the subject has been reignited. This has lead to products that are able to surveillance people for instance fall detection which then can call emergency services. One of the major advantages of radio based recognition system is that it is completely anonymous. A issue with other surveillance systems it that it has often involved cameras of some sort. This implied that peoples privacy is violated. With these new radio based system, people can be surveillanced without the persons in questions feeling like their private life is being intruded.

In this project, the question was if it was possible for a radio-based system to detect what direction a object is moving. This was to be applied in a scenario where the number of people in a room needs to be monitored. It spun from the idea limiting the amount of people inside of a building during the COVID-19 pandemic. The goal of the project was therefore to create a product to keep track of the amount of people in a room with only a radio-signal.

The goal of this document is to give a technical description of the product and also show results from tests and a discussion on the feasibility of product like this.



# 2 PREVIOUS WORK

It is a fact that humans interfere with signals produced and received by transmitters and receivers. There are several studies and articles regarding this and how to use machine learning models to extract the desired information from a disturbance. When it comes to detecting the direction of the object that caused the disturbance, which is the question in this report, there is not much information on the topic. This means that there is not much knowledge to come by on this particular product, but there are methods that can relate to this proposition.

## 2.1 Human Gesture Recognition

Something that can be argued is coinciding with direction detection is Human Gesture Recognition using wireless signals. To sense different gestures that a human does with these signals has gained an extensive amount of research attention since the large advancements in wireless technology have happened lately [1]. This is a so-called "device-free" setup since it does not require any extra devices to track the movements of the object in question.

When analyzing a received signal, the Received Signal Strength Indicator (RSSI) or Channel State Information (CSI) of the collected signal can be utilized. RSSI has been commonly used in the past. This was due to the fact that in order to extract the CSI from, for instance, a WiFi signal, expensive software-defined radios had to be used [2]. Nowadays, there are publicly available software tools that allow CSI extraction from WiFi signals [2].

When it comes to gesture recognition, RSSI shows a limited accuracy [1]. This implies that using RSSI in this context can yield unsatisfying results. CSI, on the other hand, shows an increased performance compared to RSSI when used in conjunction with different machine learning models [1].

## 2.2 Radio-Based Human Sensing

Another highly relevant thing to this project is Radio-Based Human Sensing. The task of sensing human activity is not a trivial one. Because of this, using traditional machine learning, such as logistic regression or support vector machines to name a few, is usually not sufficient enough for these tasks. Deep learning methods are, therefore a better fit to capture the depths of the features that are contained in the radio signals. Evidence for this is presented in [2].

A category of deep learning models is Convolutional Neural Networks (CNN). In [3] such a model is presented with 9 layers whose task was to recognize sign language with WiFi CSI data. It got an average accuracy of 98.01%, 98.91%, and 94.81% when testing in different environments. When conducting an experiment where the 20% of the data was mixed with different environments, the accuracy dropped to 86.67% showing a bias to the setting, rendering the model non-robust to variance in the data according to [3]. Furthermore, in [3] a Long short-term memory model (LSTM) is also presented, which was used for CSI behavior recognition. It performed with an accuracy of 90.5%, and then a modified version which yielded 97.3%. A Hidden Markov model (HMM), which is a conventional machine learning method, was also tested but gave a remarkably worse result of 73.3%.

It is apparent that deep learning models outperform the conventional machine learning methods, but they carry a drawback which is that they require significantly more data in order to be trained and yield satisfactory results [2]. There is only a limited supply of public datasets in this research field [3]. This has led to many researchers evaluating models on their own collected data, which implies that researchers have to collect the data themselves. Also, a comparison between models from different research articles can show misleading results. Furthermore, in [1] it is

concluded that, with current methodologies (especially deep learning methods), the resulting models will not be robust against dynamically changing environments.

### 2.3 Preprocessing

CSI is generally noisy and can therefore be hard to extract meaningful features in its raw form. According to [1] a low-pass filter, like a Butterworth filter, is not enough since there exists impulse noise, which has high bandwidth meaning that a low-pass filter would not produce a smooth CSI stream. Therefore does [1] suggest a principal component analysis (PCA) as an alternative. With that, most of the relevant information will be concentrated in the first components such that the rest of the components can be discarded. In [1] they also remove the first principal component (and use the the next 5). Another approach is used in [4], where a low-pass filter is used in conjunction with a PCA.

### 2.4 Features

Features extraction is often used in machine learning. When it comes to WiFi signals a common features is its frequency content. The short-time Fourier transform (STFT) shows the how the frequency evolves overtime in the signal. In [1] the discrete wavelet transform (DWT) is suggested since it reduces the amount of dimensions compared to the STFT. The DWT also provide a high frequency resolution for things at low speeds [1]. They use a 12-level DWT to decompose the 5 principal component (which they got from the PCA discussed above). Then the 5 values from the DWT is averaged. So every 200 ms they get a 27-dimensional feature vector which is then feed into their machine learning models.

### 2.5 Communication software

In the context of previous work, a special mention should be given to students taking this course in 2021. The work on their project, Detection of Static and Dynamic Indoor Movement [5], laid the foundation for the software controlling the PLUTO devices also in this project. Adaptions have been made to serve the aim of our project, but they really did the groundwork for us to build upon.

# 3 SYSTEM DESCRIPTION

This section provides the reader with an overview of the system. The functionality of the system will be described by briefly explaining all subsystems trough their structure and purpose. To see the requirements for the final system to get a even better understanding of why things are build up the way they are, please see the requirement specification [6].

## 3.1 Overview

The system is developed to detect an objects moving direction through a doorway using machine learning algorithms and Software-Defined Radio (SDR) equipment. Two Pluto SDR devices is used, where one is transmitter (Tx) and the other one receiver (Rx). The system consists of three subsystems, hardware, software and a graphical user interface (GUI). The Pluto-SDR devices together with some code for handling and saving data make up the hardware system while the machine learning (ML) models make up the software subsystem. The user interface communicates with both the software and hardware subsystem and handles the communication between the two. All subsystems were developed using python.

When the hardware subsystem has collected and stored the data, this is used by the software subsystem so train its ML-models. Those models are different types of convolutional neural networks (CNN) with different amounts of layers. Before the networks are trained however, the software subsystem preprocess the data and extract features from it. This is described further in section 5.3.

The user interface is a convenient way to maneuver the available functionalities of the detector. In the GUI a user can collect new data, evaluate pre-recorded data sets and see how different models perform. The GUI also displays relevant graphics such as a real time plot of the received signal, the last signal to be evaluated by the software subsystem, metrics from the ML algorithms such as the amount of training data it had, number of epochs trained, accuracy on validation data, F1-score and the type of model that is currently running. The GUI also displays the amount of people whom has entered or left the room according to the ML predictions since the detector started, with certainties showing on how certain the model is that it got the right answer. The user interface is covered more in depth in section 6.

To know if a passage has even happened or not, a rather simple evaluation of the variance in the signal is performed. If the variance is larger than a given threshold, the signal is sent from the hardware subsystem to the software subsystem for classification. Hence, the ML algorithms only needs to classify the direction. This is covered in depth under the hardware section 4, and more specifically in subsection 4.5.1.

## 3.2 Environment

The system was developed in a controlled indoor environment as Campus Valla in Linköping. More precisely in house B, entrance 29, room 3A:462, which is a small office with some furniture in it and a long corridor with a printer and some trash cans outside. All the relevant data collection took place at this specific location when the surrounding environment was controlled. During data collection there were never any known disturbances other than the person walking passed the detector that the system could have recorded. See figure 1 for a sketch over the environment.

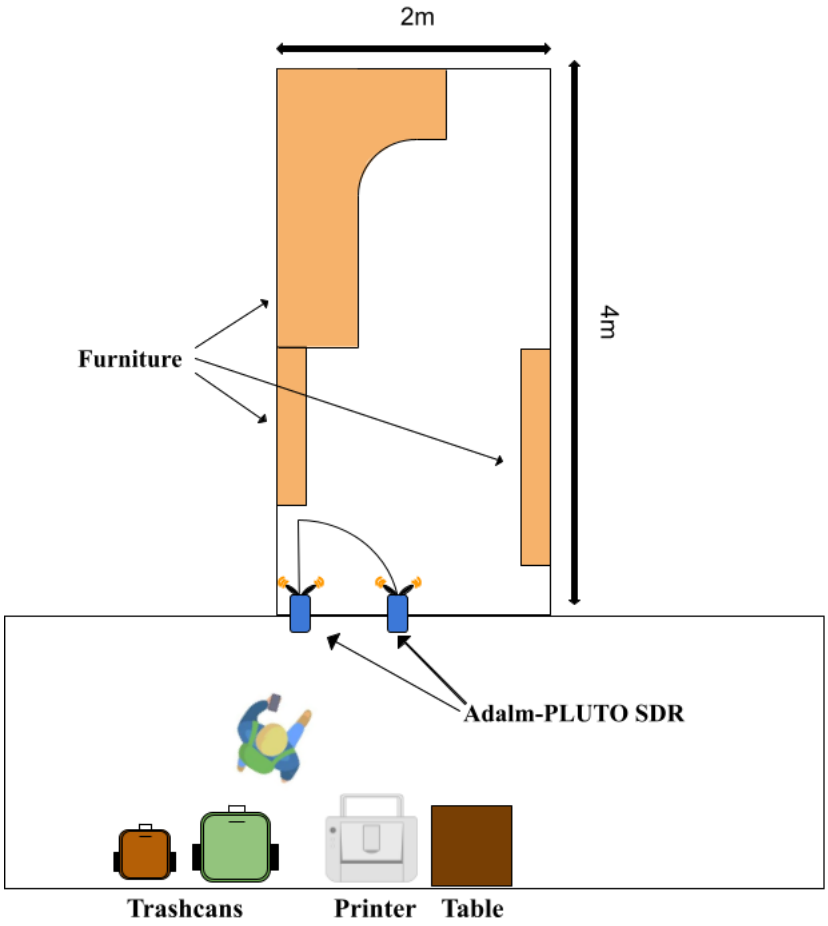



**Figure 1:** Sketch of the environment in which data collection took place.

Two Pluto-SDR devices was used and placed one on each side of the doorframe. The transmitter was placed on the right hand side of the doorframe from the perspective of standing outside of the room looking in, and the receiver was placed on the left hand side. Both devices were mounted 29cm from the ground. See picture 2.

**Figure 2:** Picture of set up used during data collection

### 3.3 Data collection

The data collection includes getting data for the classification algorithms to train on. During one data collection session, a total of $N$ number of samples set by the user will be collected. These $N$ samples are being collected in an alternating manner so that the user can walk in and out through the room while every other sample will be labeled as "in" and "out" respectively. Each data sample is five seconds long, which has been shown to be sufficiently long. See [7] for more detailed information of how data collection is set up and performed.

At the start of the project, data collection with different setups was done for ruling out setups that worked worse than others. A total of six different setups were tried, with around 100-200 data samples in each. Models were trained on the different setups and the setup seen in figure 2 was the one performing the best with the models tried. Therefore, this setup was used throughout the project.

### 3.4 Data storage

Data was structured according to schemas for the software module to easily being able to extract the labels from the collected data. Such a file containing the schemas and more valuable information about the data collections is described in a file called "description.json" and an example of the layout of this file can be seen in figure 3. More information about schemas and "description.json" can be found in 5.1.1.

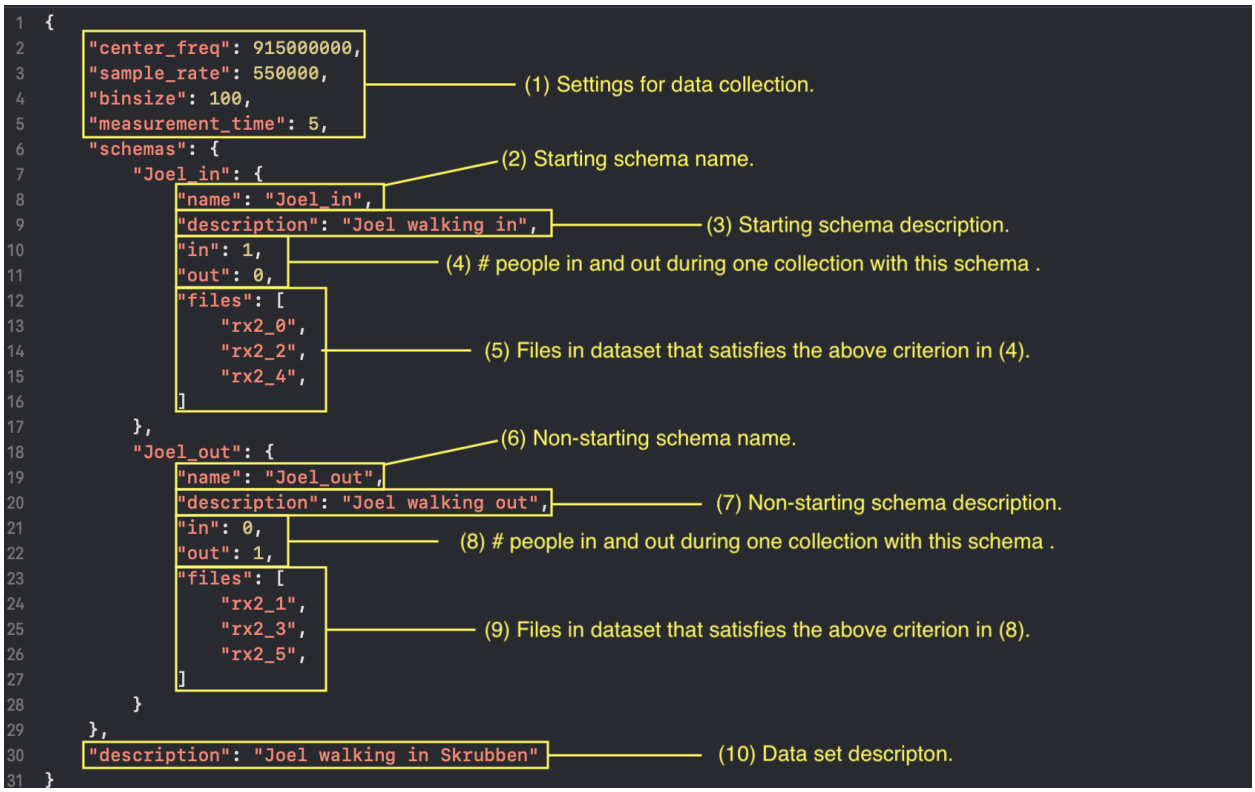



**Figure 3:** Picture to illustrate the content of *description.json*.

# 4 HARDWARE

This section describes the hardware subsystem. It consists of two ADALM PLUTO Software-Defined Radio (PLUTO SDR) devices, a host computer, and relevant software to transmit, receive and interpret signals as well as handle communication between the devices and a host computer.

## 4.1 Pluto SDR

The PLUTO devices, shown in figure 4, are easy-to-use tools to explore Software-Defined Radio and Radio Frequency Communications. The devices make it possible for the project to abstract away the analog signal processing. Instead, the data transmission can be considered a discrete memory-less channel that takes IQ symbols as input and gives distorted IQ symbols as output.


**Figure 4:** The two PLUTO devices used, transmitter Tx and receiver Rx.

The project utilizes two Pluto devices in its current configuration. One device will function as the transmitter (Tx) and another as the receiver (Rx). The devices are connected to the host computer via USB2. Each PLUTO device has the following specifications:

- Two SMA connectors for connecting antennas, Tx and Rx,
- 300 MHz - 3.8 GHz radio frequency coverage,
- 200 kHz - 20 MHz channel bandwidth, and
- USB2 port (480 Mb/s at 100% utilization) with the possible modes: network device, USB Serial device, mass storage unit.

The devices can be connected to different choices of antennas via the SMA connectors. By testing, the project decided to proceed with the antennas delivered with the Pluto devices. A change did not seem to add any benefit.

### 4.2 Host computer

As a host computer, any standard computer that fulfills the technical requirements to run programs stated in section 6 will work. During the project, the software has mainly been run on Linux-based computers.

### 4.3 Channel estimation

This subsection provides an overview of the topic. As mentioned in 2.5, the code for getting the estimated CSI was written by students taking the same course in 2021. In the technical documentation, [5] of their project, an in-depth theory description on signal propagation and channel estimation can be found.

A channel is set up between the transmitter, Tx, and the receiver, Rx. This is mathematically described by

$$y[n] = h[n]x[n] + w[n]^1 \, , w[n] \sim \mathcal{CN}(0, \sigma). \tag{1}$$

On this channel, ones (1) is sent repeatedly. When walking through the door, i.e. between Tx and Rx, the channel is disturbed, changing the CSI represented by $h$ in 1. The basic hypothesis of this project is that the changes in CSI will differ depending on whether a person walks into a room or out of it.

The hardware subsystem operates on the rolling last five seconds of the complex received signal collected from the Rx device. To prepare signal data for the software subsystem, an estimation of CSI ($h$), $\hat{h}$, is made following 2. The CSI is divided into smaller sets, bins of binsize 100. The average of each bin is calculated. The array of these bins constitutes a complex estimate of the signal CSI, $\hat{h}$, downsampled by binsize.

$$\hat{h}[n] = \frac{1}{100} \sum_{k=100n+1}^{100(n+1)} \frac{y[k]}{x[k]} \, , x[k] = 1 \tag{2}$$

The absolute value of $\hat{h}$ is also calculated, mean adjusted with the calibration mean from the channel calibration process made on system start (or when the user so chooses).

The output from the hardware subsystem to the software subsystem is the CSI, in the form of $\hat{h}$ and $|\hat{h}|$, as shown in figure 5. In the system live detection mode, the CSI information is also used by the hardware as described in 4.5.

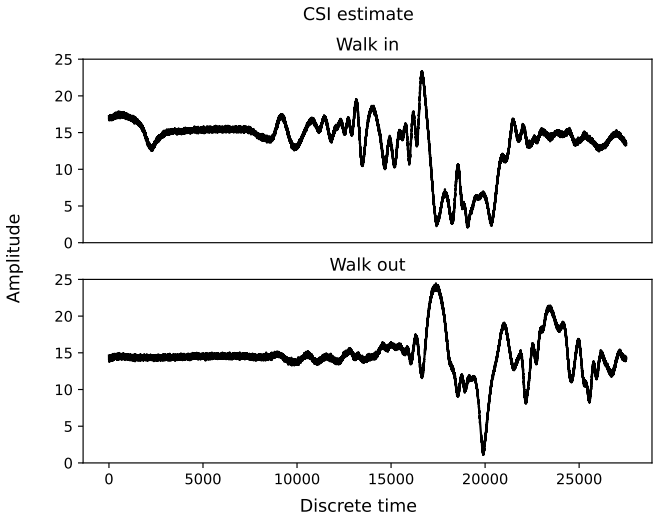



**Figure 5:** Output of hardware subsystem, 27500 estimates of the CSI.

### 4.4 Subsystem parameters

For the devices some device-specific parameters are chosen for this project. These are the center frequency, the sample rate, and the binsize. These values have been set by testing based on the code supplied by the customer.

The group has chosen the measurement time since it provides good results. A shorter measurement time would most likely be possible.

- Center frequency: 915 MHz
- Sample rate: 550 kHz
- Binsize: 100
- Measurement time: 5 s.

The parameters can not be changed using the user interface, and different settings would affect the system's performance.

### 4.5 Live detection

WaveCounter comes with two detection modes. One assisted mode, where the user tells the hardware subsystem that a person will pass through the door. In this case, the hardware subsystem waits for one (1) person to pass through and sends the CSI estimation, as described in 4.3, to the software subsystem. The other mode is a continuous mode, where WaveCounter supervises the door until the user stops the process. Every time a passage is detected, the CSI estimation is sent to the software subsystem before returning to supervise the door for the next passage.

### 4.5.1 *Passage detection*

In both of the modes described above, the hardware subsystem needs to predict whether someone has passed through the door or not before passing the CSI estimation on to the software subsystem. The ML models are only trained on predicting direction, not the absence of a passage. Passage detection can, therefore, be considered the first threshold in determining the direction of the passage.

In 4.3, the processing of the CSI estimation sent to the software subsystem is described. For this to even happened, a passage has to be detected. To do this, the hardware subsystem evaluates the variance of the CSI.

The sample buffer is used to calculate a version of $|\hat{h}|$, $|\hat{h_p}|$, downsampled to simplify calculations. The sampling frequency is somewhat arbitrary, chosen and selected through testing. The result is also passed through a low-pass Butterworth filter, described in 5.2.1, to remove high-frequency noise. From the resulting array, the values representing the 2nd second are extracted, and the variance of these values is calculated. If the calculated variance exceeds a threshold, it has proven to be an efficient trigger to detect a passage. By tests, a variance exceeding 30 has been chosen.

Note that a trigger for passage detection could be constructed in many ways. Several methods (and Python packages) can be implemented to find sudden changes in a signal. Another obvious solution, considering the project's purpose, would be to implement an ML model. Utilizing the CSI estimation variance has the benefit of being simple and has a low cost of calculation since it uses data already available.

# 5 SOFTWARE

In this section all matters concerning passage classification and data used for training related models are presented.

## 5.1 Dataset

A custom dataset $\mathcal{D}$ was created using the schema structure described in section 5.1.1. In summary, it contains data of 6 different persons passing in and out of the project lab 4508 times. *In* is defined as class 0 while *out* is defined as class 1.

The collected data, retrieved as described in section 4, consists of a 5 second 5500 Hz CSI estimation of the receiver-transmitter channel, resulting in 27500 time-samples to be pre-processed as described in 5.2. In Section 7.3 the result of the pre-processed dataset is shown by averaging over all samples.

**Definition:** Let $\mathcal{D}_{10} \subset \mathcal{D}$ be the subset such that $\mathcal{D}_{10}$ contains 10% of the samples in $\mathcal{D}$ sampled uniformly within classes, meaning a 50/50 class balance is maintained.

| Person | Num. of in | Num. of out | Total |
|---|---|---|---|
| Christian | 559 | 558 | 1117 |
| Joel | 514 | 511 | 1025 |
| Martin | 335 | 333 | 668 |
| Sebastian | 333 | 333 | 666 |
| Henrik | 183 | 183 | 366 |
| Tiger | 333 | 333 | 666 |
| Total | 2257 | 2251 | 4508 |

**Table 1:** Summary of dataset $\mathcal{D}$ which is a collection of several schemas as described in 5.1.1.

### 5.1.1 *Dataset schemas*

This chapter explains the structure of a dataset and details about data collection. This includes the concept of a schema, why they exist and how they are implemented in the WaveCounter.

A dataset is directory containing a dataset descriptor file 'descrition.json'. The descriptor file is formatted as a JSON-object with the following fields:

- *Binsize*: See section 4.
- *Center_freq*: See section 4.
- *Sample_rate*: See section 4.
- *Description*: The description of the dataset as written by the user creating it.
- *Measurement_time*: Currently unused, reserved for future improvements.
- *Schemas*: Used to label files. See detailed explanation below.

Schemas are used to label gathered files in a dataset. All schemas are formatted as JSON-objects with the following fields:

- *Name*: The name of the schema
- *In*: The number of persons going in to the room during a sample.
- *Out*: The number of persons going out of the room during a sample.
- *Description*: A short description of the samples this schema is supposed to represent.
- (*Files*): This field is only present for schemas used in a dataset. It contains the name of all files that match this schema in a given dataset.

Any given schema exist as a template in the file *schemas.json* and potentially in any number of dataset descriptors. Schema templates lack the field *files*. When creating a new schema, you actively create a new template to add to *schemas.json*.

Schemas are used in datasets to label files. When a collection is started, if any of the selected schemas are not present in the dataset descriptor, the program copies the schema template from *schemas.json* into the dataset descriptor and appends the field *files*. During a collection each file produced that matches a schema will be added to the field *files* of the schema. Note that a file can match at most one schema in each dataset.

### 5.2 Feature Preprocessing

For feature pre-processing a few tools were used. STFT was described in the design specification but was not used in the final product as it was not deemed sufficiently good. For notational simplicity, the following values were used: $f_s = 5500$ Hz, $f_{cutoff} = 100$ Hz. Other values are explained in following sub-sections. The included pre-processing tools are specified in the table below:

| Tool | Parameters | Description |
|---|---|---|
| Butterworth Low-Pass | order=4, $f_{cutoff,normalized} = \frac{2f_{cutoff}}{f_s}$ | Output size = 1x27500 |
| DWT | Haar wavelets, $I_{ms} = 250$ | Output size = 12x25, see section 5.2.5 |
| Downsample | $T = 250$, $N = 27500$ | Output size = 1x110, see section 5.2.4 |
| Min-Max normalize | - | See section 5.2.2 |
| Z-score normalize | - | See section 5.2.3 |

**Table 2:** Tools used for feature pre-processing with some parameters, they are explained more in detail by sections specified in the "Description" column.

The feature pre-processing pipeline is described below for two cases:

**1. One dimensional input** ($1 \times 110$)

1. Low-Pass filter
2. Min-Max normalize
3. Downsample

This is used for the 1DCNN, FCNN, LSTM, SVM and HMM models.

**2. Two dimensional input** ($1 \times 12 \times 25$)

1. Low-Pass filter
2. Min-Max normalize
3. DWT
4. Z-Score normalize over all $\mathbf{R}^{12\times25}$ features with empirical estimates saved from training set

This is used for the 2DCNN model.

#### 5.2.1 *Butterworth filter*

A low-pass Butterworth filter was used as a first step of pre-processing of the CSI estimates. Parameters of the filter is specified in Table 2. See result of this operation in Figure 6.

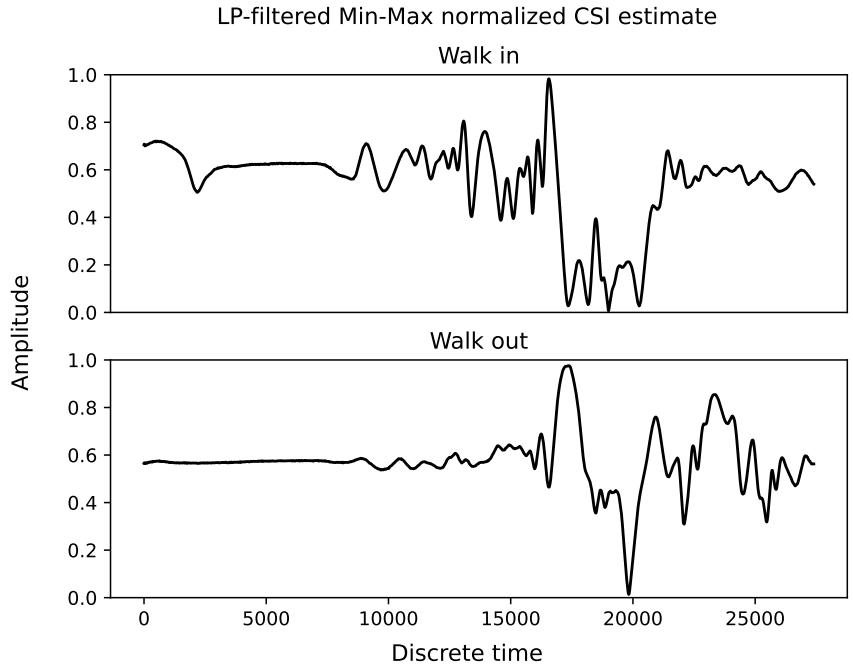



**Figure 6:** The CSI-estimates passed through the Butterworth filter specified in Table 2 after being Min-Max normalized. See Figure 5 for unfiltered CSI-estimates.

### 5.2.2 *Min-Max normalization*

The min-max normalization of $\mathbf{x} \in \mathbf{R}^{n\times 1}$, $n \in \mathbf{N}$ is $\bar{\mathbf{x}} = \frac{\mathbf{x}-\mathbb{1}_n \min_i(\mathbf{x_i})}{\max_i(\mathbf{x_i})-\min_i(\mathbf{x_i})}$, $1 \leq i \leq n$.

This makes the normalization step more invariant to changes in the channel over trials and time. For example if the bias and variance of the estimated CSI varies over time, which it has been observed to do. See result of this operation in Figure 7.

### 5.2.3 *Z-score normalization*

The Z-score normalization of $x \in \mathbf{R}$ is $\bar{x} = \frac{x-\hat{\mu}}{\hat{\sigma}+\epsilon}$, where $\hat{\mu}$, $\hat{\sigma}$ are empirical estimates of the mean and standard deviation of the underlying distribution of $x$ assumed to be normal, in this case evaluated using the train dataset when training the models (see Section 5.3). $\epsilon \in \mathbf{R}$ is added for numerical stability.

### 5.2.4 *Downsampling*

The downsampled signal $x$ of the sampled signal $X$ is

$$x[n] = X[nT] \text{ for } n \in \mathbf{Z}, \text{ constant } T \in \mathbf{N} \text{ such that } 0 \leq nT \leq N \tag{3}$$

where N is the total number of samples in $X$. See example in Figure 8.

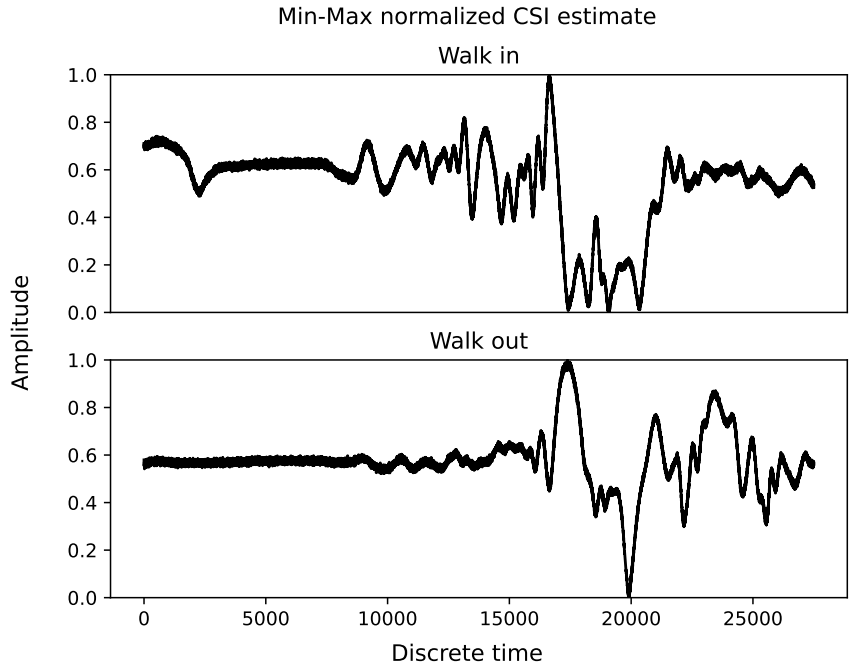



**Figure 7:** Min-Max normalization of the CSI estimates, seen in Figure 5. Note that it's simply transformed to values between 0 and 1, while all samples have the same proportions relative to neighbours in time.

### 5.2.5 *DWT*

A 12-level DWT (Discrete Wavelet Transform) with Haar wavelets was performed on time-segments $[nI_{ms}, (n+1)I_{ms}]$ over the signal, $n \in \mathbf{Z} \geq 0$, $I_{ms}$ a time interval in milliseconds, converted to interval of samples $I_n$ by $I_n = I_{ms} f_s / 1000$. The resulting 12 detail-components of each time-segment were concatenated into a two-dimensional representation $I_{DWT} \in \mathbf{R}^{12, T_{DWT}}$ of the signal $X$, where $T_{DWT} = \frac{N}{I_n} = \frac{27500}{1100} = 25$ as seen in Figure 10. In Figure 9 the DWT has also been Z-score normalized.

### 5.2.6 *STFT*

The STFT performs the DFT for time-segments of a signal, generating a two-dimensional representation of the signal over frequency and time as seen in Figure 11. There was not enough frequency content in the collected signal $X$ to extract a valuable STFT, resulting in a poor representation of the signal and bad prediction accuracy why it was not used.

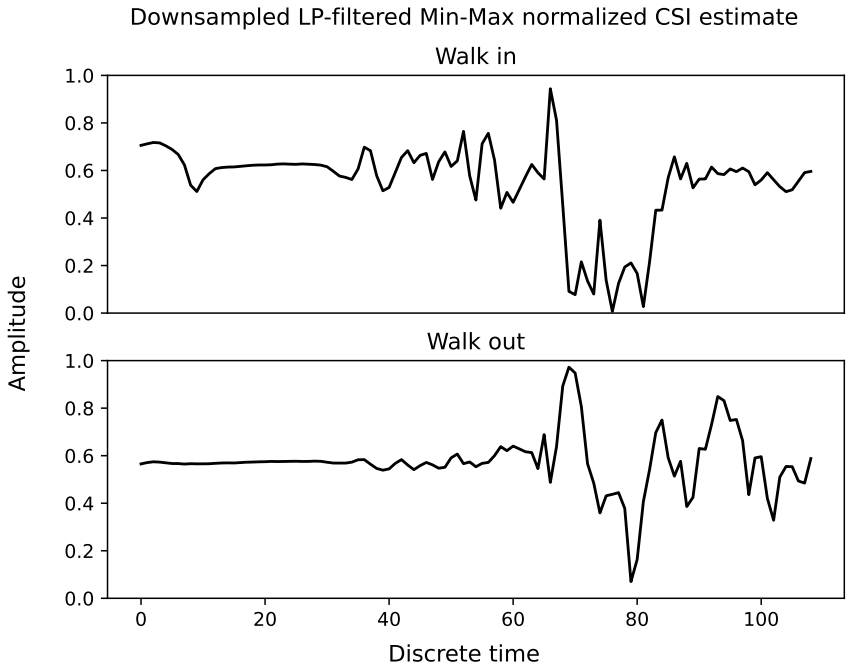



**Figure 8:** Downsample from 27500 to 110 of the Min-Max normalized and Butterworth filtered CSI estimates, seen in Figure 6.

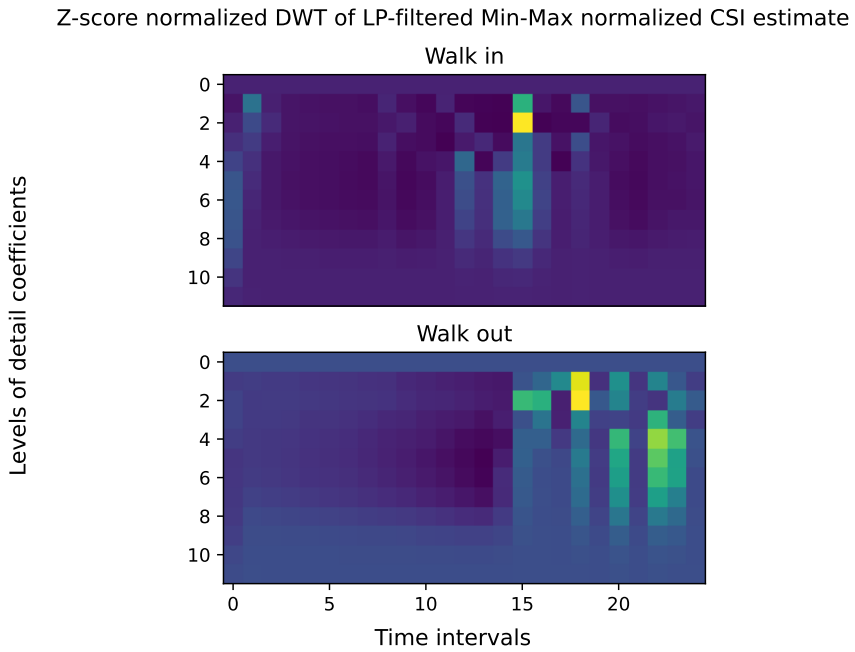



**Figure 9:** DWT of 27500 sized Min-Max normalized and Butterworth filtered CSI estimates, seen in Figure 6, giving a $1 \times 12 \times 25$ sized input for models with 2D input. Not that in the Figure the DWT has also been Z-score normalized, see 5.2.3 and also compare with Figure 10.

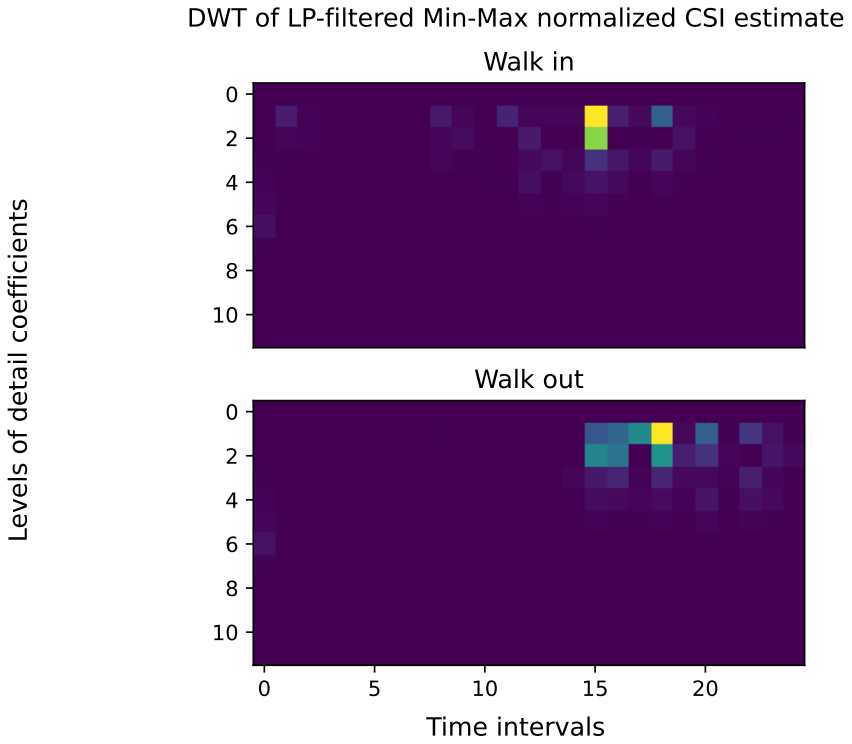



**Figure 10:** DWT of 27500 sized Min-Max normalized and Butterworth filtered CSI estimates, seen in Figure 6, giving a $1 \times 12 \times 25$ sized input for models with 2D input. In the Figure the DWT has not been Z-score normalized, compared to in Figure 9.

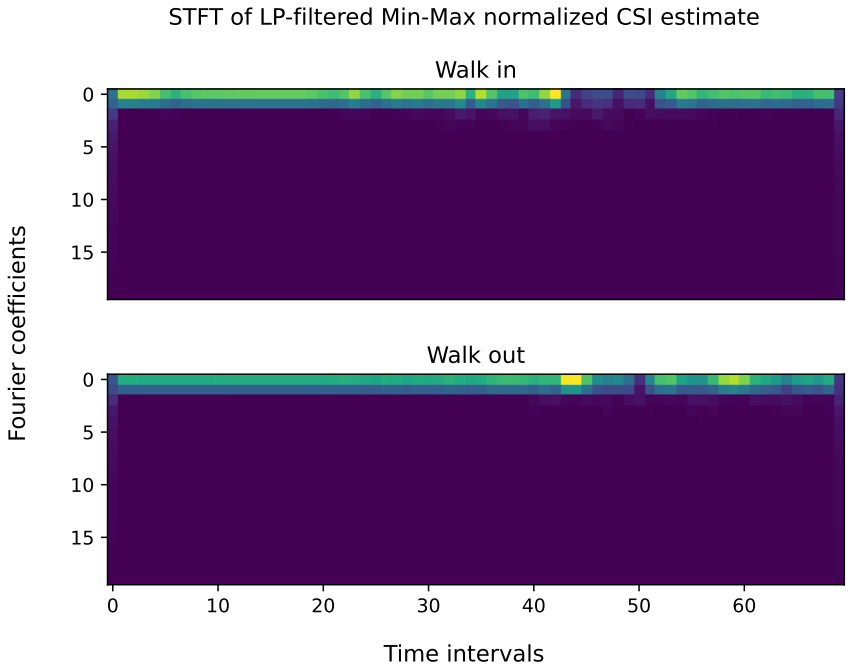



**Figure 11:** STFT of 27500 sized Min-Max normalized and Butterworth filtered CSI estimates, seen in Figure 6.

## 5.3 Models

Out of the described models in the design specification not all were used in the final product, as specified below

- **Used models:** 1DCNN, 2DCNN
- **Unused models:** LSTM, FCNN, SVM, HMM

The 1DCNN and the 2DCNN gave satisfactory performance, why the other models were not investigated further. However, an evaluation of the performance of all neural networks and the SVM and HMM is presented in section 7.2.1.

### 5.3.1 *Performance metrics*

The following notations are important to understand the performance metrics:

- **TP**: True positive, samples that had a positive (1) class and were classified as such.
- **FP**: False positive, samples that had a negative (0) class but were classified as positive (1).
- **TN**: True negative, samples that had a negative (0) class and were classified as such.
- **FN**: False negative, samples that had a positive (1) class but were classified as negative (0).
- **#** "number of ".

**Accuracy** is defined as,

$$\text{Accuracy} = \frac{\#TP + \#TN}{\#TP + \#FP + \#TN + \#FN}. \tag{4}$$

**Recall** is defined as,

$$\text{Recall} = \frac{\#TP}{\#TP + \#FN}. \tag{5}$$

**Precision** is defined as

$$\text{Precision} = \frac{\#TP}{\#TP + \#FP}. \tag{6}$$

With the recall and precision, the **F1 score** is defined as

$$\text{F1-score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{7}$$

An F1-score = 1 means a perfect performance of the model. Looking at the F1-score and not only accuracy is important since an accuracy of 1 can be misleading if the dataset is unbalanced in classes.

### 5.3.2 *Neural network training*

All neural networks were trained using the following:

- **Optimization algorithm**: Adam, learning rate = 0.0001, $\beta = (0.9, 0.999)$, $\epsilon = 1e-08$, weight decay=0
- **Loss function**: Negative Log Likelihood

- **Batch size**: 4
- **Dataset**: $\mathcal{D}$ with a 80/20% train/test split for the models in the product with performance listed in Table 9 in section 7.2.1. $\mathcal{D}_{10}$ with a 75/25% train/test split for the models with performance listed in Table 8 in section 7.2.1.

### 5.3.3 *1DCNN*

The architecture of the 1DCNN is specified in Table 3. Performance can be read in Table 8 in section 7.2.1.

| **Layer** | **Parameters** |
|---|---|
| Input | Size $1 \times 110$ |
| Conv1D | 125 filters, kernel size 30, stride 1 |
| ReLU | - |
| Maxpool 1D | Max of each filter from Conv1D output |
| Fully connected | 125 in, 128 out |
| ReLU | - |
| Fully connected | 128 in, 2 out |
| Softmax | - |

**Table 3:** Architecture for the 1DCNN used in the product.

### 5.3.4 *2DCNN*

The architecture of the 2DCNN is specified in Table 4. Performance can be read in Table 8 in section 7.2.1.

| **Layer** | **Parameters** |
|---|---|
| Input | Size $1 \times 12 \times 25$ |
| Conv2D | 1 in-channel, 6 out-channels, kernel size 5, stride 1 |
| ReLU | - |
| Maxpool 2D | Kernel size 2, stride 2 |
| Conv2D | 6 in-channels, 16 out-channels, kernel size 3 |
| ReLU | - |
| Maxpool 2D | Kernel size 2, stride 2 |
| Flatten | - |
| Fully connected | 64 in, 120 out |
| ReLU | - |
| Fully connected | 120 in, 84 out |
| ReLU | - |
| Fully connected | 84 in, 2 out |
| Softmax | - |

**Table 4:** Architecture for the 2DCNN used in the product.

### 5.3.5 *FCNN*

The architecture of the FCNN (Fully Connected Neural Network) which was tested, but not included in the product, is specified in Table 5. Performance can be read in Table 8 in section 7.2.1.

| **Layer** | **Parameters** |
|---|---|
| Input | Size $1 \times 110$ |
| Fully Connected | 110 in, 300 out |
| ReLU | - |
| Fully Connected | 300 in, 100 out |
| ReLU | - |
| Fully Connected | 100 in, 2 out |
| Softmax | - |

**Table 5:** Architecture for the FCNN tested, but not included in the final product.

### 5.3.6 *LSTM*

The architecture of the LSTM (Long short-term memory) which was tested, but not included in the product, is specified in Table 6. Note that the LSTM is bi-directional. Performance can be read in Table 8 in section 7.2.1.

| **Layer** | **Parameters** |
|---|---|
| Input | Size $1 \times 110$ |
| Bi-LSTM | 110 in, $128 \times 2$ out |
| ReLU | - |
| Fully Connected | 256 in, 512 out |
| ReLU | - |
| Fully Connected | 512 in, 256 out |
| ReLU | - |
| Fully Connected | 256 in, 128 out |
| ReLU | - |
| Fully Connected | 128 in, 2 out |
| Softmax | - |

**Table 6:** Architecture for the LSTM tested, but not included in the final product.

### 5.3.7 *SVM*

A SVM with a radial-basis-function kernel was fitted and trained automatically using scikit-learn 1.1.3 [8] on input of size $1 \times 110$. The SVM model was not used in the final product but an evaluation of its performance can be found in Table 8 in section 7.2.1.

### 5.3.8 *HMM*

Two HMM's (Hidden Markov Models) were trained exclusively on data from class in(0) and from class out(1) respectively using expectation maximization. Input size was $1 \times 110$. To classify a single sample as in or out, the likelihood

of the sequence was calculated for each trained HMM. The HMM that gave the highest likelihood classified the class of the sample as its own. The HMM's and expectation maximization was implemented using the Python package hmmlearn 0.2.8 [9]. This algorithm with a double HMM was not used in the final product but an evaluation of its performance can be found in Table 8 in section 7.2.1.

# 6 USER INTERFACE

The user interface consists of a webapp, constructed in the framework Svelte and run by a Flask-server inside a Docker-container. This section aims to explain some details of the User Interface that was skipped in the User Manual for brevity and clarity. Therefore carefully read the User Manual before reading this section.

## 6.1 Home

The home page is fully explained in the User Manual [7]. Please read it for an explanation of how the Home page works.

## 6.2 Live

This section aims to give some details on certain parts of the live page that was skipped in the User Manual [7]. To understand the basic functionality look at the User Manual [7].

### 6.2.1 *Metrics*

Upon selection of a model, the interface displays six different metrics associated with the model. These metrics are:

- *Type:* The underlying type of the model. *1DCNN* and *2DCNN* means 1- or 2-dimensional Convolutional Neural Network. See section 5 for a detailed explanation these terms.
- *Datasets:* The number of different datasets used when training this model.
- *Epochs:* The number of times the model used each dataset during training.
- *Samples:* The number of samples used to train this model. Each epoch the model has evaluated all of these samples.
- *Accuracy:* The achieved accuracy of the model on test data. See section 5 for an explanation of how this score was calculated.
- *F1-Score:* The achieved F1-Score of the model on test data. See section 5 for an explanation of how this score was calculated.

### 6.2.2 *In and Out counter*

These counters display the predictions done by the model. When evaluating on gathered test data, the precision and recall for each of the two classes is displayed based on the labels of the files. Note that this evaluation is not done using cross-validation, and therefore the F1-score displayed might not match the F1-score of the model. Using any other button in the Interface during evaluation is not recommended and results in undefined behavior of the counters!

When evaluating on live gathered data the counters display the confidence the model had in its latest prediction as a percentage. The counter that was last incremented will be colored slightly green to clearly indicate in which direction the last passage was.

### 6.2.3 *Evaluating on datasets*

Each folder in the directory 'data' containing a file named 'description.json' will be considered a dataset and displayed in the User Interface. DO NOT create your own folders in this way and attempt to use them to collect data; instead read the instructions in the User Manual on how to collect data and create datasets. To move datasets to a new computer, copy the entire dataset folder to the other computer. It is recommended to reduce the file size using a compression tool before attempting to move them. Datasets might be large and it might take considerable time to copy them otherwise.

When selecting 'evaluate', the program goes through each dataset in the same order that they were selected. All schemas are extracted from the dataset and the files attached to the schema will be read and a prediction produced for each file. Every 50 predictions and at the last prediction the interface will be updated with current information about the evaluation. There are two caveats to be aware of during this process: 1, while the program will go through the datasets in order, this is not guaranteed for the schemas and 2, 'recall' is usually undefined for one class at the start of the evaluation: this is because no files belonging to the class has yet been evaluated and therefore the recall for that class is undefined.

### 6.2.4 *Plots*

There are two plots displaying a processed version of the received signal. The left plot updates continuously one time per second as the system is running and displays the received signal. The right plot displays the 5 seconds long signal sent to the software system for the latest prediction of direction.

Both plots use the estimate of absolute value of $h$ as described by Equation 4.3. Before display the estimate is filtered using a Butterworth filter as described in 5.2.1, and finally downsampled by picking every hundredth sample.

## 6.3 Flask

The backend API is served by a Flask-server, and is responsible for handling of schemas, datasets, collecting of data and running live predictions on models. All communication between the modules go through the server.

## 6.4 Docker

The docker container provided in the Wavecounter is a lightweight, stand-alone software package which includes everything needed to run the application. This container is isolated from your computer and any other containers you might be running, except for it having access to specific files and directories inside the main folder and exposing port 8000 and 3000 to the host computer. Thus, keeping the interference with the host system as minimal as possible.

The build settings for the container running the Wavecounter can be found in the files *Dockerfile* and *docker-compose.yaml*. These files are designed to copy required files to the container. To modify the container for your particular use case we refer you to the documentation for Docker:

# 7 RESULTS

To evaluate how good the WaveCounter performs in a live scenario some tests were carried out. How well the different models performs on the collected data for training/testing was also computed to get a sense of which performance to expect from the product in a live scenario. In this section the results from these tests will be presented.

## 7.1 Passage detection

Passage detection, described in detail in 4.5.1, can be considered the first step in determining the direction of a passage. For a passage to even be classified by the models in 7.2.1, passage detection must first be triggered. The system requirements state that the product should detect a passage with an accuracy of 75 percent.

### 7.1.1 *Detection performance*

Detection performance was tested in a live situation with a number of individuals passing through a door at random. In every test 50 detections were collected in total, also noting the number of misses in each test. The results are presented in Table 7.

| Test | Misses | Accuracy |
|---|---|---|
| Test 1 | 0/50 | 100% |
| Test 2 | 2/52 | 96% |
| Test 3 | 6/56 | 89% |
| Test 4 | 6/56 | 89% |

**Table 7:** Results on live tests for passage detection.

### 7.2 Passage classification

The models described in 5.3 have been tested on subsets of $\mathcal{D}$, as presented in this section. First results of models trained on $\mathcal{D}_{10}$ are presented to compare all models, even those not included in the product. Secondly results of the primary models 1DCNN and 2DCNN trained and tested on $\mathcal{D}$ are presented, to show a more true accuracy and F1-score.

#### 7.2.1 *Model performance*

All models were trained and tested on $\mathcal{D}_{10}$ with a 75/25% train/test split to give an idea of how they compare. The top performances for the models are found in Table 8. These were trained and tested 9 times, one time per number of epochs in [10, 20, 30, 40, 50, 60, 70, 80, 90, 100] .

| Model name | Accuracy | F1 Score |
|---|---|---|
| 1DCNN | 80.4% | 0.771 |
| 2DCNN | 82.1 % | 0.773 |
| FCNN | 58.0% | 0.514 |
| LSTM | 60.7% | 0.545 |
| SVM | 61.6% | 0.538 |
| HMM | 65.2% | 0.435 |

**Table 8:** Top performances when trained and tested on $\mathcal{D}_{10}$ with a 75/25% train/test split.

In Table 9 the models used in the product are trained for 100 epochs on $\mathcal{D}$ with a 80/20% train/test split.

| Model name | Accuracy | F1 Score |
|---|---|---|
| 1DCNN | 91.1% | 0.91 |
| 2DCNN | 81.3% | 0.81 |

**Table 9:** Performance when trained and tested on $\mathcal{D}$ with a 80/20% train/test split for 100 epochs.

### 7.3 Dataset

In Figure 12 the average over $\mathcal{D}$, Min-Max normalized and Butterworth filtered is seen. While, as seen in Figure 5 the output is caothic, there is indeed a pattern when looking at the average over the entire dataset. The same averaging has been applied to Z-score normalized DWT of the Min-Max normalized Butterworth filtered CSI, seen in Figure 13. While the distinction between classes is perfect for averaging over $\mathcal{D}$, it is not a completely fair picture which is demonstrated when averaging over the lower and upper half of $\mathcal{D}$ respectively.

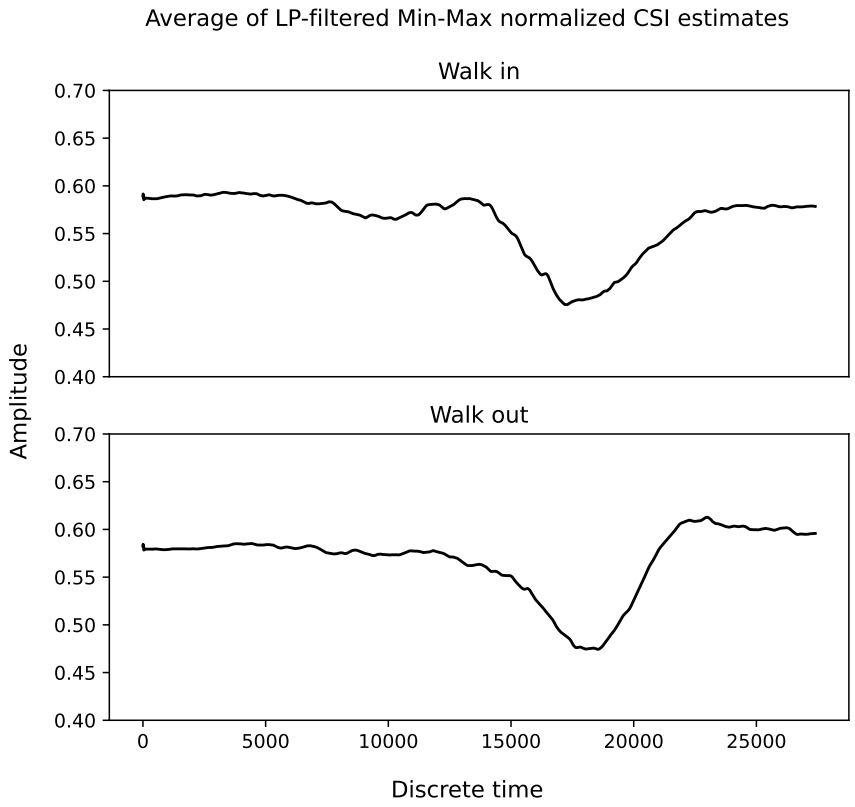



**Figure 12:** Average (over subsets of $\mathcal{D}$) Min-Max normalized Butterworth filtered CSI.

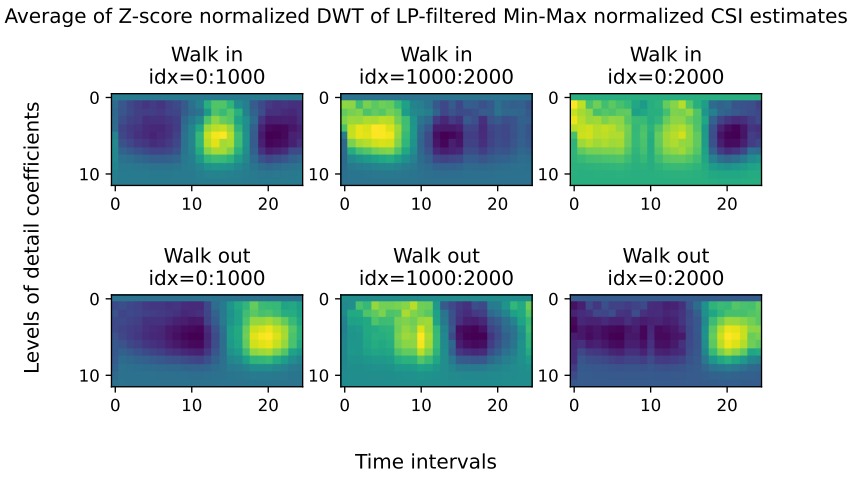



**Figure 13:** Average (over subsets of $\mathcal{D}$) DWT with Z-score normalization on Min-Max normalized Butterworth filtered CSI.

# 8 DISCUSSION

In this section, a discussion about the WaveCounter will take place. The discussion will be about results, possible future work, and limitations the system currently have. Active choices made during and different routes to explore for further product development will also be discussed.

## 8.1 General

- Considering that the product only uses two PLUTO devices, which are set up in a symmetrical manner relative to each other, the system performs very well. One would expect the ML models to have a tough time distinguishing "in" and "out" because of the symmetric setup, but this clearly is not the case. So is it possible to detect a human moving direction only using two SDR devices, CSI, and ML - it absolutely is!
- The product does not generalize very well with respect to the environment. If this is because of the models used or because of the training data is difficult to say, but collecting more data in different environments would be good for reliability reasons.

## 8.2 Hardware

- In the product, two PLUTO devices were used. One with the role of a transmitter, and one with the role of a receiver. Adding one or more receivers could make the product more robust since there is more and more varied data could be gathered. One possible setup would be to put the extra receiver inside the room.
- In the setup used the PLUTO devices were set up in a doorframe on a height of 29cm, pointing directly toward each other. It is not certain that this is the optimal setup when using two devices. It might be better to put them on ground level or maybe at breast height. It might also be beneficial to not have them pointing directly at each other. At the beginning of the project, some slight experimentation on this subject was done, but a conclusion that the setup used, see 2, was sufficient enough was reached. This is worth exploring further for future work.
- There have been some observations that the performance of the system decays with time. The suspicion is that the Pluto devices become warm after a few minutes of usage, which should increase measurement noise. This means that data samples collected at the end of a collection series will be noisier than the ones at the start as well as the system requires cooling if a long run time is needed.

## 8.3 Software

- In section 5.3.2 it is made clear that the models used in the product are trained on $\mathcal{D}$ with a 80/20% train/test split. This does not give the best model since training on 100% of $\mathcal{D}$ would have given the model a better generalization. However, the product also requires that every trained model has a test accuracy associated to it, thus a portion of $\mathcal{D}$ must be reserved for the test.
- For all models, including the ones not used in the product, a thorough hyper-parameter optimization was not performed such as number of layers, nodes etc. The selection of number of layers and nodes was rough and based on previous works as described in section 2 and specifically in the Design specification [10]. Most likely better performance could have been found by optimizing these hyper-parameters, however, since the chosen

hyper-parameters gave a good performance and achieved accuracy above what was required (see Requirement specification [6]) this was not investigated further.

- While testing the product it was made clear how sensitive it is to changes in the physical environment and surrounding equipment. During our training, the corridor outside the designated room (room 3A:462) was organized in one way, sketched in figure 1. During some regrouping of ISY, the copy machine overnight moved from one side of our door of interest to the other. A big metal frame table was added to the environment. Suddenly the models made a lot of wrongful predictions. Putting everything back in its previous position, the models started delivering great results again. This is a good lesson in the sensitivity of the models, but also the need to train on a lot of varied data.
- As can be seen in table 9 it is clear that the best performing model is 1DCNN, were it is almost 10 units higher than 2DCNN. This shows that the feature extraction used, more specifically the discrete wavelet-transform (DWT), which the 2DCNN utilized, did not yield a better result. So for this particular case of radio-based recognition it is not sufficient with both frequency and time domain features but instead works better with just the time domain information.

## REFERENCES

[1] 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, no. 10, pp. 98–104, 2017.

[2] 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.

[3] S. K. Yadav, S. Sai, A. Gundewar, H. Rathore, K. Tiwari, H. M. Pandey, and M. Mathur, "Csitime: Privacy-preserving human activity recognition using wifi channel state information," *Neural Networks*, vol. 146, pp. 11–21, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608021004391

[4] 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)*. IEEE, 2017, pp. 967–974.

[5] R. Mannberg, M. Andersson, E. G. Emma Beskow, J. Nilsson, G. Suihko, and J. Qu, "Detection of static and dynamic indoor environments, technical report."

[6] H. Ahlinder, S. Andersson, M. Dahl, C. Gustavsson, J. Henneberg, and T. Kylesten, "Detection of an object movement direction indoors: Requirement specification, 2022."

[7] ——, "Detection of an object movement direction indoors: User manual, 2022."

[8] "scikit-learn," https://scikit-learn.org/stable/index.html, 2022.

[9] hmmlearn. [Online]. Available: https://hmmlearn.readthedocs.io/en/latest/

[10] R. Mannberg, M. Andersson, E. G. Emma Beskow, J. Nilsson, G. Suihko, and J. Qu, "Detection of static and dynamic indoor environments, design specification."