# System Design Specification

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

December 16, 2022

Version 1.0

Status

| | | |
|---|---|---|
| Reviewed | Martin Dahl | 2022-10-07 |
| Approved | Danyo Danev | 2022-10-31 |

## 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 |
|---|---|---|---|---|
| 1.0 | 2022-10-06 | Finalized version for customer. | Project group | Martin Dahl |

# 1 INTRODUCTION

The product is a system using software-defined radio to count the number of people entering and exiting a room, thus keeping track of the number of people in the room. This document provides a design specification, and possible solutions, for the system design, which will serve as a foundation for continued work.

# 2 SYSTEM DESCRIPTION

The system consists of a hardware part consisting of several Software Defined Radio (SDR) units described in section 3, a software part consisting of the human detection algorithms described in section 4 and a GUI part described in section 5. An overview of the modules and the communication and data flow between them can be seen in figure 1.

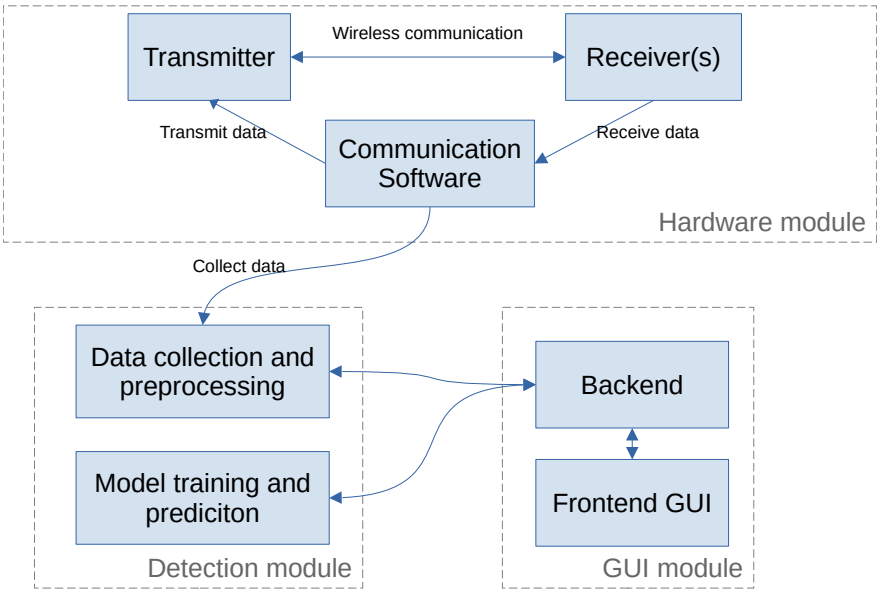


**Figure 1:** System overview. Lines indicate data flow and/or control signals

# 3 HARDWARE

The hardware module will consist of two Pluto SDR devices and software for the communication between the host computer and the Pluto devices. There will be one transmitter device and one (or more) receiver devices. Each Pluto device has the following specification:

- Two SMA connectors for instrumentation and antennas,
- 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 device

The output from the hardware subsystem is going to be the estimated CSI, which is going to be further processed by the software subsystem. All code used for getting the estimated CSI was written by students taking the same course in 2021. In their System design specification [1] everything one needs to know about the signal propagation and channel estimation can be found.

### 3.1 Communication software

The hardware module comes bundled with software for controlling and collecting data from the Pluto devices. The controlling software is supplied by the orderer and the project group will not put substantial resources in further development of this software, except what is needed to make it compatible with this projects interests and GUI. The data collection is handled by the communication software but only in short-term memory. A detailed description of persistent storage can be found in section 3.3.

### 3.2 Data collection

The data collection will consist of a gathering of training, validation, and test data. During a session, the system will be set up and disturbance will be done during the recording. The disturbance to the signal will be zero or more people that moves in or out of the door that is located between the transmitter and receiver(s). Each data set will contain multiple samples of data that each needs to be labeled. When the signal is labeled it will be processed in a way so that features all ML-models will benefit from having will be extracted. Features only used by some ML-models will instead be extracted by the software subsystem. In figure 2 a flowchart of the data collection is shown.

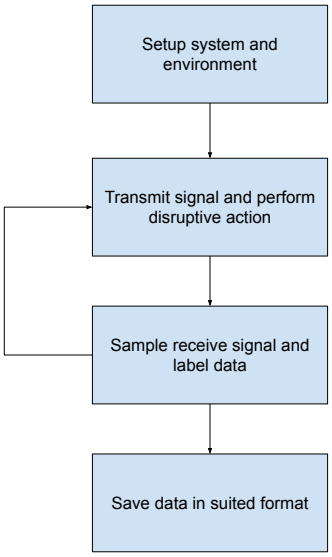


**Figure 2:** The flowchart describes the data collection.

### 3.3 Data structure

The structure of the collected data sets will be important to make training and locating the correct data sets for training efficient. The project will therefore have a directory called 'data' with subdirectories for each data collection occasion. These subdirectories will be named after the date and occasion during which they were recorded, e.g '220910_01'. Each data sample will consist of two files in the required directory, and these are e.g for the first (number 01 data collection) 'rx2_abs_01.iq' and 'rx2_01.iq'. This way, the data will have a nice and convenient structure and specific data sets will be easy to find. In each data set, there will exist exactly one json-file containing:

- hardware specifications such as center frequency, binsize and sample rate for the Pluto SDR
- "description": A optional short description of the dataset.
- "measurement_time": The length in seconds of signals in this dataset.
- Label for signals stored in the dataset. The labels should at least include:
  - A descriptive name of what is gathered
  - "who": Which of the project members that passed trough the system
  - "in": #People are going into the room in this signal
  - "out": #People walking out of the room in this signal.
  - "data" A list of the stored files that fit this label.

Existing labels will be preferred over creating new ones. Labels will be shared between datasets, e.g. the same label can appear in one or more datasets referring to different files. In the beginning of the project, the 'in'- and 'out' parts of a label will only take binary values, and only one can be '1' at a time. Later in the project data collections with more than one thing happening at each data collection will be explored. Data will be divided into train-, test- and validation-data at a later stage. In figure 3 a flowchart of the data structure is shown.

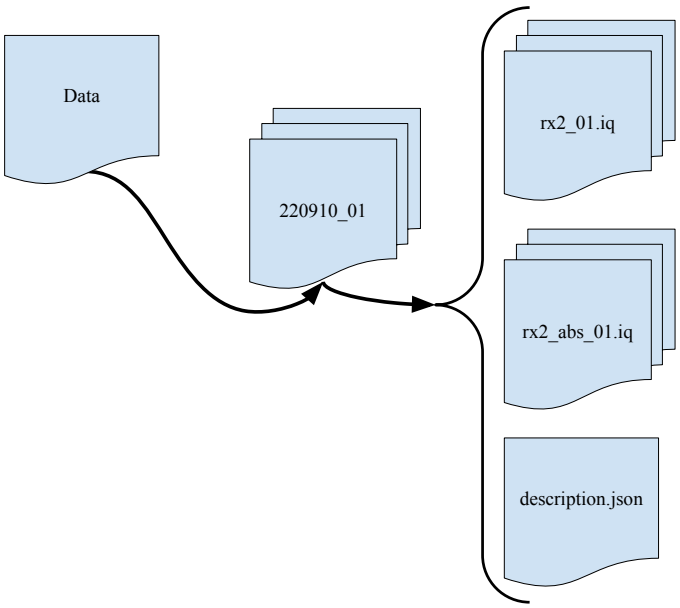


**Figure 3:** The flowchart describes the data structure.

### 3.4 Environment

All data collection will take place in the same controlled and fixed lab environment. The location for the data collection will be a small office at ISY (the department of Electrical Engineering) at the university of Linköping. This is a office the project group has gotten access to by the customer to facilitate development of the product. Since other projects also have access to this room some small things in the room might change between data collections, otherwise the environment is as controlled as can be.

During all data collections the transmitter will be located on one side of the door-opening. Two receivers will be located on the other side of the door: one on the inside of the room and the other on the outside. If this proves to be successful, the experiment will be retried using only one receiver. More receivers might be added to the project later if the collected data using two receivers proves to be insufficient for training the machine learning models mentioned more in section 4.2. Where the added Pluto SDR devices will be located relative the door frame will be decided first when the group decides that more SDR devices are needed.

# 4 SOFTWARE

The software module is responsible for the collection, preprocessing, and extracting of relevant features from the data, training, validation, and testing of the machine learning modules. The software subsystem will be written in Python 3 using machine learning libraries and frameworks such as PyTorch and scikit-learn. A flowchart describing the software subsystem can be seen in Figure 4.

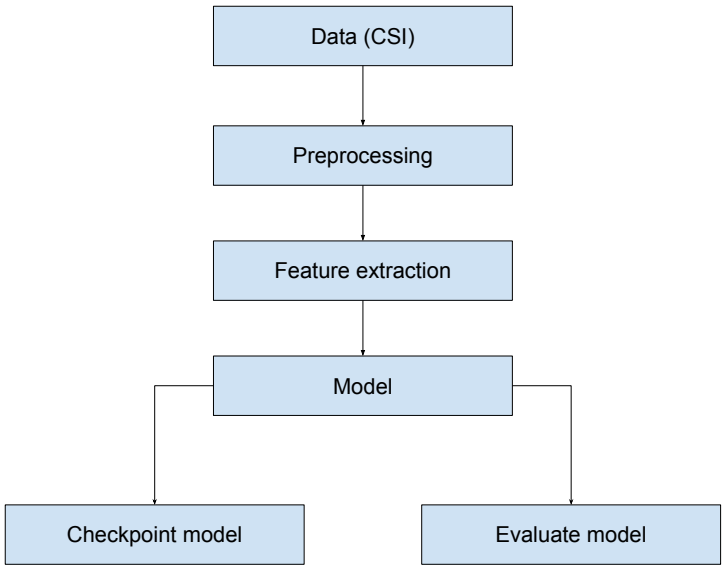


**Figure 4:** The flowchart describes the software subsystem.

## 4.1 Data and Feature Extraction

### 4.1.1 *Data from hardware measurements*

CSI is in general very noisy making it hard to make sense of it or extract valuable features. This means that the signal needs to be de-noised in some way. Noise is often in the higher frequencies so a conventional low-pass filter could be suitable. However, in CSI there can exists impulse noise (noise with peaks that deviates a lot from the original signal), and since they have high bandwidths, a low pass filter cannot produce a smooth CSI stream [2]. Nevertheless, [3] did use a de-noising method with both a Butterworth and weighted moving average filter before applying PCA (described below).

A more viable option here is principal component analysis de-noising [2]. Principal component analysis (PCA) is a technique usually used for dimensional reduction. It transforms the data into another coordinate system. In this coordinate system the axis, which are the principal components (PC), are chosen based on the variance in the data. The first principal component (PC1) is align to the data where there is the greatest variance. The second principal component (PC2) is chosen where there is the second greatest variance, and so on until there are no dimensions left.

Commonly does the first bunch of principal components capture 99% of the variance in the data and thus the rest can be discarded, resulting in dimensionality reduction. Most of the information of the signal is therefore captured in those first principal components, which indicates that the noise is residing in the higher order components [4]. By removing those the data becomes more clean and less redundant. In another approach the first component is discarded, and the next five are kept, discarding the remaining higher order components [2]. Both variants will be tested and compared.

#### 4.1.2 *Feature Extraction*

For simple models such as SVM and HMM feature extraction is required. A simple method of extracting features is using the discrete STFT (Short-time Fourier Transform). For 2D-CNNs the STFT can also be good since the STFT generates an image of the signal. However, the number of frequency components used from the STFT can be selected to better suit the SVM and HMM models [2]. DWT (Discrete Wavelet Transform) is another feature extraction method which can be argued to be preferred over STFT since it also decreases the number of dimensions [2]. In [2] roughly the following procedure is used: A 12-level DWT is used to decompose the five principal components extracted during noise reduction. The values of the DWT are averaged. Every 200 ms a 24-dimensional feature vector is extracted using the energy in each wavelet level (averaged) and the difference between each levels between consecutive 200 ms intervals.

### 4.2 Machine Learning Models

For the project, a set of different ML models are described. Mapping radio signals to human activities is a complex task since the waves can reflect on other people and objects in the surrounding environment [5]. In human sensing applications, deep neural networks have shown to be useful. A deep neural network is an artificial neural network with many layers of interconnected neurons. In hybrid models, issues with a single model can sometimes be addressed by using a combination of ML models.

The choice of models described in the section is somewhat arbitrary but is based on previous experience and several articles within the field of human sensing using radio waves.

#### 4.2.1 *Support Vector Machine (SVM)*

A Support Vector Machine (SVM) is a linear supervised machine learning model taking input $\mathbf{x} \in \mathbf{R}^d$ to, in this case, predict the binary output $y \in \{1,-1\}$. We let $\hat{y}$ denote the predicted output, given by $\hat{y} = g(\mathbf{w}^T\mathbf{x} + b)$ for some weight vector $\mathbf{w} \in \mathbf{R}^{d\times n}$, bias $b \in \mathbf{R}^n$ and activation function $g(x)$. A common activation function is given in Equation 1.

$$g(x) = \begin{cases} 1 & \text{if } x >= 0 \\ -1 & \text{if } x < 0 \end{cases} \tag{1}$$

The goal is to find $\mathbf{w}$ and $b$ such that the hyperplane that the parameters define separate classes in the data by a maximum margin, which is possible if the classes are linearly separable. Since data rarely has linearly separable classes (case A), the general case (case B) with non-linearly separable classes is described. In Figure 5 an example is shown with case A. In case B, a hyperplane is found such that as few samples as possible are misclassified, this is achieved with the minimization objective in Equation 2 with regularization included.

$$\text{minimize } \lambda||\mathbf{w}||^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x} + b))\right] \tag{2}$$

Equation 2 can be understood as follows: If $1 - y_i(\mathbf{w}^T\mathbf{x} + b) <= 0$ we have that $y_i(\mathbf{w}^T\mathbf{x} + b) >= 1$ which means that $y_i$ and $\mathbf{w}^T\mathbf{x} + b$ have the same sign, as well as the sample being past the hyperplane level 1. This will give a correct classification $\hat{y}$ using $g(x)$ as defined in Equation 1. In this case the separator is good for the given sample. In the opposite case there is a sample inside of the margin which is penalized.

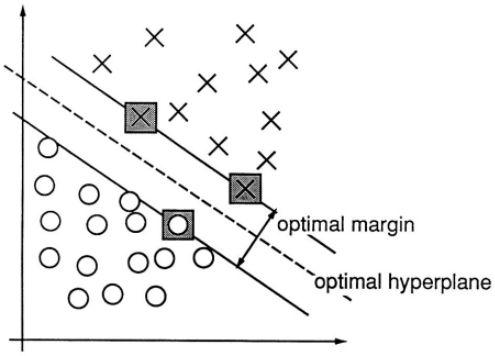


**Figure 5:** Hyperplane separating classes by a maximum margin. Taken from [6].

Note that the separating hyperplane is linear, which will have great issues separating non-linear data. For example with a 2 dimensional feature set, one class is inside of a circle, the other on the outside. See Figure 6. Directly applying a SVM will add a line cutting through the circle. This can be fixed using the "kernel-trick" where instead of training the SVM on $(x_1, x_2)$ it is trained on $(x_1^2, x_2^2)$. In general, the input feature vector $\mathbf{x}$ is transformed using the kernel function $\kappa(x_i, x_j) = \phi(x_i)\phi(x_j)$ where $x_i, x_j$ is any combination of features from the same sample.

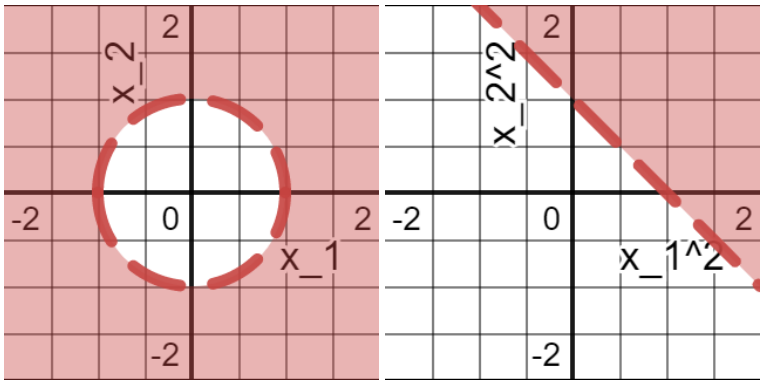


**Figure 6:** Demonstration of transforming features to handle non linearly separated classes. On the left: $x_1^2 + x_2^2 > 1$, on the right $v_1 + v_2 > 1$ where $v_1 = x_1^2, v_2 = x_2^2$.

### 4.2.2 *Hidden Markov Model (HMM)*

HMM aims to model an non-observable(hidden) markov process $Y$ by observing the process $X$ influenced by the outcome of $Y$, only discrete time processes are considered. Specifically, the relation in Equation 3 applies.

$$\mathbf{P}(X_n \in \mathbf{A} | Y_1 = y_1, ..., Y_n = y_n) = \mathbf{P}(X_n \in \mathbf{A} | Y_n = y_n) \tag{3}$$

where $\mathbf{A}$ is some set.

This means that any value in the process $X$ is only dependent on the state of the markov process $Y$ at the same time step. Furthermore, these conditional probabilities depend on transition probabilities $\mathbf{P_T}$ between states in the markov process $Y$ and emission probabilities $\mathbf{P_E}$ for a state in the markov process $Y$ to generate a state in the observed process $X$. $\mathbf{P_E}$ and $\mathbf{P_T}$ are the parameters of the HMM.

A HMM is commonly used to to (non-exhaustive) [7]:

1. Calculate the probability of an observed sequence given the parameters of the HMM.
2. Maximize $P(X|\mathbf{P_E}, \mathbf{P_T})$, with respect to the set of parameters $\mathbf{P_E}, \mathbf{P_T}$. In other words, the HMM is trained to best fit the observed sequence.
3. Calculate the most probable hidden sequence given the observed sequence and the parameters of the HMM.

For a classification task, the first point is useful. One HMM is trained per class, and the HMM giving the highest probability for a sequence (time-series) will give the class. However, it is also possible to have the hidden states be the classes and classify a time series on majority vote. In that case you only train one HMM and use the third point. The parameters of a model, considering the second point, is found using the expectation maximization algorithm.

### 4.2.3 *Fully Connected Neural Network (FCNN)*

A fully connected neural network consists of layered super positioned linear models separated by activation functions. A neural network is fully expressive, meaning it can approximate any continuous function. It is trained using back-propagation and some descent method. Backpropagation is an algorithm to perform the chain rule which is a good tool for heavily nested functions. An example of a descent method is Gradient Descent, where the gradient is given by backpropagation. However, accelerated variants of it such as ADAM is commonly used. Different activation functions can be used to avoid problems such as vanishing gradient, exploding gradient, linearity. It's common to use the leaky-ReLU to avoid these issues.

The Neural network can be used for regression or classification depending on the final layer. For classification, the final layer is ended with a softmax or sigmoid activation function, depending on the number of classes and the relationship between classes. If regression is wanted, the activation function can be omitted.

### 4.2.4 *Convolutional Neural Network (CNN)*

The CNN takes the FCNN a step further with an inductive bias assuming a correlation between neighboring elements in the data. This is useful when the data is a time series, images, or any similar data. In the case of CSI data, one can apply a CNN on spectrograms, as described later in this chapter. The network trains filters and performs convolution in every layer. The CNN is ended with a fully connected layer to perform the final prediction.

### 4.2.5 *Recurrent Neural Networks (RNN)*

RNN are designed to work on sequential predictions by utilizing a feedback mechanism in each recurrent unit. The context of the prediction will affect the prediction by introducing a concept of memory to neural networks. RNNs are prone to two issues while training deep neural networks, vanishing, and exploding gradients.

### 4.2.6 *Long Short Term Memory Neural Network (LSTM)*

LSTM is a more robust version of recurrent neural networks, RNN, capable of learning long-term time dependencies while addressing issues with training deep neural networks. The architecture uses a hidden state, passed down the chain of cells of the neural network as short-term memory, making it last as long as possible.

LSTM has a chain-like structure, as seen in Figure 7. The key to the architecture is the cell state, running along the entire chain with only some minor interactions. It's easy for information to flow unchanged along the chain.

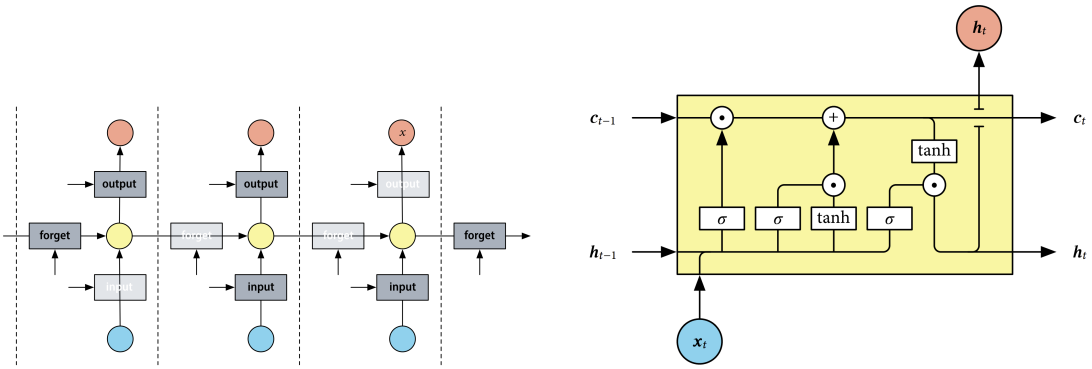


**Figure 7:** Information flow of an LSTM and an LSTM cell. Attribution: Geoffrey Hinton, Chris Olah (CC BY 4.0)

The LSTM cells can remove or add information to the cell state using a gate feature. Each cell has three gates (input, forget, output). The gating mechanism associated with LSTM can be described by Figure 8

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} \odot \begin{bmatrix}0\\1\\1\\0\end{bmatrix} + \begin{bmatrix}5\\6\\7\\8\end{bmatrix} \odot \begin{bmatrix}1\\0\\0\\1\end{bmatrix} = \begin{bmatrix}5\\2\\3\\8\end{bmatrix}$$

$$\boldsymbol{h}_{t-1} \quad \boldsymbol{g} \quad \boldsymbol{x}_t \quad 1-\boldsymbol{g} \quad \boldsymbol{h}_t$$

**Figure 8:** Gating mechanism for LSTM. Attribution: Marco Kuhlmann (CC BY 4.0)

The gates can let information pass through the cell or add and remove information from the cell state. The gates are composed of a sigmoid function and a pointwise multiplication operation. The sigmoid layer outputs a number between zero and one, describing how much of each component to let through.

The LSTM architecture's strength is maintaining long-range connections and recognizing the relationship between values at the beginning and end of a sequence. This generally makes for good results, even if it comes at the cost of being computationally costly. In classification tasks, it is most efficient if there is a small set of classes.

Besides the regular LSTM, there is also the bidirectional LSTM. The BiLSTM adds one LSTM layer with the reversed direction of information flow. This allows the neural network to utilize information from both ends. There are also other specialized flavors of the LSTM besides the basic vanilla versions.

#### 4.2.7 *Machine Learning for CSI classification*

A review has been made of using machine learning for CSI classification. For each model described in this section an approach is suggested. Most papers reviewed show a clear pattern of using deep models such as LSTM and CNN, since classic models are not complex enough for CSI data. Models will be implemented in Python scikit-Learn [8] and PyTorch [9].

- **SVM**: Is a classic machine learning model and should be tested before proceeding to more complex models. SVMs have been seen to be outperformed by RF and HMM [2]. Furthermore, SVMs have been stated to not be sufficiently expressive for the task of CSI classification [10]. Before usage feature extraction and noise reduction is required [2]. Results show that RBF kernels are preferred in terms of performance, but may be slower to train than linear kernels [11]. A SVM will be tested.
- **HMM**: HMM has been applied with success to classify CSI data, achieving higher performance than a SVM model. Noise reduction and pre-processing is required [2]. A HMM will be tested.
- **FCNN**: A traditional neural network (FCNN) is used in [3] for predictions on CSI data with satisfactory performance. MLPs, which are equivalent to FCNNs are also mentioned in [5]. However, these references were the only ones found to be using a FCNN whereas most use LSTM or CNN models, as seen in points below. A FCNN will be implemented.
- **CNN**: 1D-CNNs have been used frequently to classify CSI data [12], [13]. It is also possible to use 2D-CNNs applied on spectograms as described in [5] and implemented in [14]. A 1D-CNN will be implemented.
- **LSTM (RNN)**: Several papers implement LSTMs (as well as a bi-directional one) with good performance for predictions on CSI data [2], [14], [10]. In [2] a LSTM outperforms a HMM to little surprise. An ordinary LSTM will be implemented, but not a bi-directional one.

### 4.3 Model Training and Testing

The dataset will be divided into three separate sets, training, validation and testing data. The training data will be used in order to train the model and will be repeatably be used as such. An epoch is when the model has gone through the whole training dataset once. The performance that the model gets on the training set will continue to increase after each epoch. But this will specialise the model to the training set, resulting in a overfitted model. A overfitted model is a model that have learned to be really good on the training set, but consequently becoming terrible on any other dataset that the model has never seen before. For this purpose and to get a fair measurement on how well the model is preforming, the validation set will be used (which is unknown to the model). So after each epoch the model will be tested on the validation set to get a more realistic measure on how well the model is preforming.

With the validation data, tuning of the hyperparameters are done until a desirable outcome is achieved. But since the parameters are chosen based on the models performance on the validation set the results may be biased towards that particular dataset. So to get a proper final measurement of the performance of the model, one that is not biased in any way the third set, the test set (which is also unknown to the model), is used to get the concluding score of the model. The evaluation of the model on the test data will be after the last epoch is finished.

So the training data is used for training the model, validation data is used for tuning hyperparamters and mitigating overfitting and lastly the test data is used to get a final score of the performance of the model.

The data will be split into 60 % training data, 20 % validation data and 20 % test data. The data will be spilt by taking random samples from the whole dataset. The sampling needs to be random in order to avoid getting unwanted biased. Taking random samples will usually end in a balanced dataset (provided that the whole dataset is balanced). So no further action to keep the balance in the datasets are needed.

### 4.4 Model Evaluation

The models will be evaluated based on often they produce prediction that is correct. The model evaluation will only take data samples from the test data set. The data that the model will be trained and evaluated on will always have a person that disturbs the signal by moving either right or left. This means that it will be binary classification problem. For binary problems there are *TP* (True Positive), *FP* (False Positive), *TN* (True Negative) and *FN* (False Negative). *Positive* and *negative* denotes what prediction the model gave (positive could be 1 and negative 0). *True* and false signify if the prediction that the model constructed was correct or not. So for example *TP* means a prediction of *positive* that was *true* (correct). From this it is possible to construct a confusion matrix, see table 1.

| / | Ground truth | |
|---|---|---|
| Prediction | # TP | # FP |
| | # FN | # TN |

**Table 1:** Confusion matrix where # means "number of". So # TP is the number of true positives.

The confusion matrix gives a visualization of how well the trained model preforms. These scores can then be used to calculate a score based on the models performance. Since the data will be balanced (same amount of "in-" and "out-passage" data samples) the F1-score is suitable. The F1-score is the harmonic mean of the *recall* and the *precision*. The *recall* is defined as,

$$Recall = \frac{\#TP}{\#TP + \#FN}, \tag{4}$$

which means that it is the fraction between correctly identified classification of a class and how many instances of the class there was. This means that the *recall* is a measure on how sensitive the model is. The *precision* is defined as

$$Precision = \frac{\#TP}{\#TP + \#FP}, \tag{5}$$

so it is the fraction between correct classification of a class and all the classification of the particular class. So a measure on how many of the classifications were correct. With these the F1-score can be defined as

$$F1 - score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}. \tag{6}$$

# 5 USER INTERFACE

The user interface is the interface from which the system is controlled. The user may use the interface to calibrate the hardware parameters, collect and store data, train and run the classification algorithms, and compare classification results of different algorithms [15]. The user will also be able to see the channel logs, the bit error rate (BER), and a system log. A mock-up of the user interface is shown in Figure 9.


**Figure 9:** A mock-up of the user interface, displaying buttons (black) and different windows.

## 5.1 Software and frameworks

The UI will consist of a backend and a frontend. The frontend will be a web application created using the framework svelte. The frontend will communicate with a backend running a flask-server with the ability to launch sub processes running communication code and ML-algorithms.

## REFERENCES

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

[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, no. 10, 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)*. IEEE, 2017, pp. 967–974.

[4] B. Li, "A principal component analysis approach to noise removal for speech denoising," in *2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, 2018, pp. 429–432.

[5] 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.

[6] C. Cortes and V. Vapnik, "Support vector networks," 1995.

[7] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

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

[9] "Pytorch," https://pytorch.org/, 2022.

[10] A. Zhuravchak, O. Kapshii, and E. Pournaras, "Human activity recognition based on wi-fi csi data -a deep neural network approach," *Procedia Computer Science*, vol. 198, pp. 59–66, 2022, 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks / 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921024509

[11] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin, "Training and testing low-degree polynomial data mappings via linear svm," *Journal of Machine Learning Research*, vol. 11, no. 48, pp. 1471–1490, 2010. [Online]. Available: http://jmlr.org/papers/v11/chang10a.html

[12] 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

[13] X. Shen, Z. Ni, L. Liu, J. Yang, and K. Ahmed, "Wipass: 1d-cnn-based smartphone keystroke recognition using wifi signals," *Pervasive and Mobile Computing*, vol. 73, p. 101393, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574119221000523

[14] P. Fard Moshiri, R. Shahbazian, M. Nabati, and S. A. Ghorashi, "A csi-based human activity recognition using deep learning," *Sensors*, vol. 21, no. 21, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/21/7225

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