Frankfurt University of Applied Sciences

–Faculty of Computer Science and Engineering–

"Tiny Machine Learning for Ultrasonic Object Classification: A Naive Bayes Approach on Red Pitaya

Bachelor Thesis

Bachelor of Engineering (B.E.)

presented by

Nguyen Tien Anh Ha

Matriculation numbber: 1356973

Referent	:	Prof. Dr.-Ing. Peter Nauth
Korreferent	:	M.Sc. Julian Umansky

DECLARATION

I hereby certify that I have written this report independently and no sources other than those specified in the bibliography used.

All passages that are literally or analogously from published or still taken from unpublished sources are identified as such made.

The drawings or illustrations in this work are mine been created or provided with a corresponding reference.

This work is not in the same or a similar form in any other examination authority has been submitted.

Frankfurt, 12. October 2023

Nguyen Tien Anh Ha

ABSTRACT

The rapid advancements in autonomous technologies have highlighted the importance of accurate object classification based on material properties. Ultrasonic sensors, with their ability to offer cost-effective and reliable detection, have emerged as a promising tool in this domain, especially in applications like autonomous vehicles and security systems.

Our primary objective in this study is to apply the capabilities of ultrasonic sensors for the precise classification of objects based on their inherent material properties. In the end, a novel Smart Sensor Application have been developed that not only simplifies the machine learning pipeline but also provides the flexibility to remotely upload and modify models. This adaptability ensures that the system remains updated with the latest classification parameters and techniques.

The sensor system prototype, integrated into car seats, is designed to predict human presence, focused on the differentiation between materials, particularly human and non-human entities. This standalone system can work without the need for external UDP software, promptly providing feedbacks upon detecting specific material properties.

Our approach employs Fourier Transform techniques combined with filters to extract unique frequency signatures from ultrasonic waves. These features vary distinctly based on the material they reflect off, enabling accurate classification. Additionally, we delve into the Naïve Bayes Classifier algorithm, achieving a remarkable 90% accuracy with a small dataset of 2,000 samples on the Red Pitaya platform using Tiny Machine Learning. By integrating ultrasonic sensing with machine learning, we present a promising direction for future developments in autonomous systems.

Keywords: Ultrasonic sensing, autonomous systems, Naïve Bayes Classifier, Standalone Sensor System, Tiny ML

ACKNOWLEDGMENTS

First, I would like to express my utmost gratitude to my supervisor, Prof. Dr. Peter Nauth, who gave me the opportunity to work on this fascinating project on the topic of "Object classification using ultrasonic signals." I greatly acknowledge his invaluable advices and suggestions on various aspects of my research and writing.

I would like to give thanks Mr. Julian Umansky for his assistance with the Redpitaya sensor board.

I also want to express my sincere thanks to the German Academic Exchange Service (DAAD) for offering me an Exchange Scholarship, which provided me with the unique opportunity to conduct my thesis in Germany, without which it would have been impossible for me to carry out this research.

Finally, without the encouragement and assistance of my friends, as well as the ongoing support of my family, I would not be able to complete my thesis on time.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ADC Analog to Digital Converter

FFT Fast Fourier Transform

GPIO General Purpose Input Output

GNB Gaussian Naïve Bayes

GUI Graphical User Interface

HMI Human Machine Interface

MAP Maximum A Posteriori

NN Neural Network

PCB Printed Circuit Board

PDF Probability Density Function

RP Red Pitaya

TOF Time of Flight

UDP User Datagram Protocol

US Ultrasonic Sensor

1 INTRODUCTION

1.1 Motivation

Over the past few decades, object classification using ultrasonic sensors has become a key research area in autonomous systems. Most ultrasonic-based systems consist of numerous sensors placed all around the vehicle [1]. The distance between the sensor and an obstacle is determined by measuring the time of flight(TOF) between the transmitting of the ultrasonic wave and the received corresponding echo. By processing the TOF data from multiple sensors simultaneously, the precise location of the obstacle can be mapped in a two-dimensional plane centered on the vehicle. This map facilitates advanced functionalities, including automatic parking, obstacle-triggered braking, and distance reporting to the user. To enhance the safety features of autonomous applications, the design of systems that utilize intelligent sensors capable of distinguishing diffrent object materials has been proposed. Ultrasonic sensors are well-known for their object detection capabilities due to their reliability - less than 1 cm in distance measurements of up to 6m [2]. Ultrasonic sensors can effectively address complex problems related to object recognition of both stationary and moving objects [3]. Building an object detection and differentiation system using an ultrasonic sensor and an embedded system is more cost-efficient and efficient than approaches that employ computer vision techniques [4].

Therefore, in this project, we analyzed and evaluated the implementation of tiny machine learning on an embedded system for object discrimination. The ultrasonic sensor SRF02 was chosen for the classification task due to its numerous advantages. Furthermore, we studied the implementation of a sophisticated Naive Bayes model to enhance the classification accuracy of the system.

1.2 Objectives

The goal of this project is to investigate the integration of Machine Learning technique with the current ultrasonic sensing system to achieve accurate and reliable different object materials. The system can detect, measure distance and distinguish between surfaces based on analysis of the reflected wave's frequency spectrum.

Currently, software developed by Mr. R. Michalik facilitates class differentiation using a box classifier, though its performance could be improved. Therefore, we would like to explore more sophisticated machine learning algorithms that can be implemented on the Red Pitaya system. In doing so, we must un-

derstand the system's current stage and familiarize ourselves with the Naive Bayes Classifier in particular as well as general Machine Learning principles.

Automation of the entire process is also a vital aspect of our research. Originally the ultrasonic sensor was started by the UDP client but now will be programmed to start automatically at the reboot. In summary, our objectives are:

- To identify the possible challenges and limitations of using ultrasonic sensors for object classification
- To develop a prototype of a human-machine interface with button and LED for sensor feature collection, eliminating the need for external client software.
- To implement the Naive Bayes Classifier as a probabilistic classifier technique on Red Pitaya for real time object classification
- To create a machine learning pipeline that can be easily modified and updated with new models

1.3 Outline of the document

The thesis is divided into six chapters, each covering different aspects of the study. Chapter 1 introduces the main focus of this study, stating the objectives and providing a brief overview of the study which primarily focuses on ultrasonic sensing, the Naive Bayes algorithm, and enhancements to the current sensor system's learning phase.

Chapter 2 delves into the specifics of ultrasonic wave characteristics and Gaussian Naive Bayes theory. The chapter also discusses the practical challenges and issues related to these systems.

Chapter 3 focuses on the design and implementation of the button interface for the learning phase. It also provides a comprehensive guide on integrating the Naive Bayes algorithm into the system.

Chapter 4 reveals the software implementation of the system, including the data collection and processing phases. The chapter also introduces the new Smart Sensor Application and its functionalities

Chapter 5 presents the experimental results and a detailed analysis of these outcomes, highlighting the effectiveness and limitations of the proposed approach.

The research concludes with Chapter 6, where we summarize the findings of the work and discuss about future research directions.

2

BACKGROUND

All the techniques, concepts and components used in this research are introduced in this chapter. It provides a basic explanation of how ultrasonic sensors operate as well as the various pre processing methods used throughout this study.

2.1 Ultrasonic Sensors

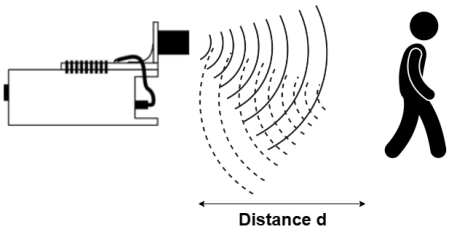


Figure 2.1: Ultrasonic Sensor System

An ultrasonic sensor is typically composed of two main components: trasmitter and a receiver. These two components act as transducers, convering one form of energy into another. The transmitter generates an high frequency ultrasonic wave from electrical energy and emits it into the environment. When this wave encounters an object, reflects back towards the sensor, where the receiver captures it and convert it back into Voltage signal. Then, a computer or other embedded system interprets this signal to derive useful information from the measurements. In industrial application, after transmitting ultrasonic waves, the sensor also works as a receiver than can capture reflected acoustic signal

Ultrasonic sensors can be further divided into two groups: Electrostatic and Piezoelectric, depending on the physical construction and working principle of the sensor.

- Piezoelectric transducers work based on piezoelectric effect, a characteristic of some materials such as crystals, certain ceramic that allows them to expand or contract when an electric field is applied. An ultrasonic wave is created when a voltage is applied to a piezoelectric material, which causes a mechanical deformation. Furthermore, when this wave is reflected back and encounters the object, it causes a mechanical strain that generates a corresponding voltage, transforming the acoustic energy back into an electrical signal.

- In contrast, electrostatic trasducer working principle is similar to capacitors. They are made up of two plates that are divied by a dielectric or

an insulator. One of these plates vibrates in response to an ultrasonic wave, changging the space between plates.As a result, The difference of the distance between two plates varies its capacitance. This change in capacitance is used to analyze the received signal.

The time of flight(TOF) , which measures how long does it takes for an ultrasonic pulse to travel to an object and go back to the transducer, together with the speed of sound in the medium(typically air). It can be used to determine the distance between the object and the ultrasonic sensor. This relationship is defined by the following function:

$$d = \frac{1}{2} c_{\text{sound}} \Delta t \tag{2.1}$$

where,

- d is the object distance,
- c_{sound} is the speed of sound in air,
- Δt is the time of flight (TOF).

2.2 Signal Processing

2.2.1 Fast Fourier Transform

To examine frequency domain feature, Fourier Transform is typically used to convert a signal from time domain to frequency domain. In this case, the frequency spectrum of the ultrasonic signal is analyzed to classify the object. The ultrasonic signal is sampled at a rate of 200 kHz, which is the maximum sampling rate of the Red Pitaya. Therefore, the signal consists of samples using ADC converter. When signal is converted to a discrete value using an A/D convertor, the Discrete Fourier Transform(DFT) can be used to convert discrete signal to frequency domain using the following equation:

$$X(e^{j\omega}) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} \tag{2.2}$$

In equation 2.2, the sampled version of collected data is referred to $x(n)$ and N is determined by the closest number to the window.The widow width N in this study is set at 8192.

2.2.2 Features Extraction

The feature selection is one of the most important tasks during classification. While some features can be extracted directly from samples of raw data, other need to be calculated or taken from the preprocessed data. After transforming raw data into frequency domain using FFT, 10 features are used to extract

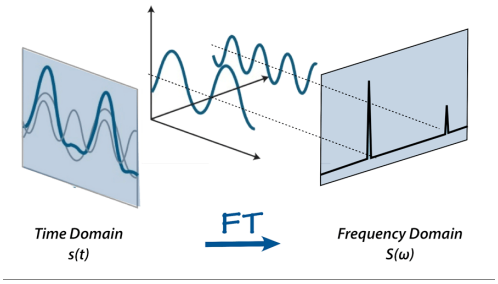


Figure 2.2: Overview of Fast Fourier Transform

the characteristic of the object. Moving Average (MA), Moving Average of Difference (MAD), and Moving Variance (MV) are the three main function that those features primarily rely on. The general equation form of these functions are illustrated in Table 2.1.

Table 2.1: Feature Equations

Features	**Equation**
Moving Average	$\frac{1}{N}\sum_{i=t-N+1}^{t} x_i$
Moving Average of Difference	$\frac{1}{N}\sum_{i=t-N+1}^{t} (x_i - x_{i-1})$
Moving Variance	$\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$

Algorithm 1: Moving Average Calculation

Input : A filter $*f$ and input value *val_in
Output : Updates the filter $*f$ with new values and the calculated moving average
Function `runMeanFilter`(f, *val_in*):
  sum $\leftarrow$ sum $-$ buffer[idx] $+$ val_in
  buffer[idx] $\leftarrow$ val_in
  idx $\leftarrow$ (idx $+ 1$) mod idx_max
  filter_val $\leftarrow$ sum/idx_max

The Moving Average calculates the average of specific number of data points within a sliding window that moves along the data stream. Algorithm 1 starts by determining the difference between the previous and the current data from the `"sum"` variable, with data from the current position of the `"buffer"`. This data is then stored in the next position of the buffer. The counting variable `"idx"` increases by taking the remainder of the division of `"idx"` and `"idx_max"`, ensuring that `"idx"` resets when it counts up to `"idx_max"`, thus creating a sliding window function. Finally, the average value is calculated by dividing the `"sum"` by the total number of values `"idx_max"`, and the final filtered value is stored in the `"filt_val"` of the struct `"filter_t"`.

Algorithm 2: Moving Average of Difference

Input : A filter $*f$ and input value $*val\_in$
Output : Updates the filter $*f$ with new values and the calculated average difference
Function `RunDiffFilter(`f, val_in`):`

```
new_diff ← |latest_val − val_in|
sum ← sum + new_diff - buffer[idx]
buffer[idx++] ← val_in
idx ← (idx + 1) mod idx_max
lastest_val ← val_in
filter_val ← sum / idx_max
```

On the other hand, the Moving Average of Difference (MAD) calculates the average rate of change over a period. In Algorithm 2, the input arguments are similar to the regular Moving Average filter, with the addition of a `"new_diff"` variable. `"new_diff"` calculates the absolute value between the previous and the current data measured from a feature. Unlike the one mentioned in Algorithm 1, this `"new_dif"` variable compares the current data with the data in the `"buffer"`. Then, the difference is added to the `"sum"` value. Subsequently, the `"new_diff"` value is stored in the next position of the `"buffer"`, while the `"latest_val"` takes the `"val_in"` pointer value in. In addition, the `"idx"` continues counting up, and the average value is periodically generated and returned as the `"filter_val"`.

Last but not least, Algorithm 3 shows the working principle of Running Variance (MV) function. This function only requires the `"filter_t"` as the input. The Function is followed the variance equation that was mentioned in Table 2.1.

Algorithm 3: Variance Calculation

Input : A filter $*f$
Output : The variance value
Function `runVarianceFilter(`f`):`

```
sum ← 0;
for idx = 0 to idx_max − 1 do
    diff ← f.buffer[ix] − f.filt_val;
    sum ← sum + diff²;
variance ← sum/idx_max;
return variance;
```

2.3 Machine Learning

2.3.1 Overview

The challenge of classifications is figuring out which category the new input data belongs to. Generally speaking, machine learning algorithms can be separated into two groups: supervised and unsupervised. The supervised learning

algorithm makes predictions based on a set of samples [5]. Since the data set had been labelled, each feature will belongs to specific class. On the other hand, unsupervised learning does not have labels attached to them. The goal of an unsupervised learning algorithm is to create different techniques for grouping data and clustering data into multiple groups. In machine learning world, the

Table 2.2: Comparison of various classification algorithms [6]

Algorithm	Error	Computational Cost	Memory Requirements	Difficult to implement
Nearest Neighbour	Med-Low	High	High	Low
Decision trees	Medium	Medium	Medium	Low
Linear least squares (LS)	High	Low	Low	Low
Neural Networks	Low	Medium	Low	High
Support vector machines (SVM)	Low	Medium	Low	Medium
Naive Bayes	Medium	Low	Low	Medium

Naive Bayes algorithm stands as a testament to the power of probabilistic approaches, based on Bayes' theorem. This algorithm has significant attention due to its wide-ranging applications across diverse domains, from the intricate task of spam email detection to the emotional recognition. One of the advantages of the Naive Bayes algorithm lies in its computational efficiency and simplicity, making it particularly adept at handling voluminous datasets [7][8] . However, this strength is also a weakness by its inherent assumption of feature independence, which can occasionally render it unreliability for tasks where inter-feature relationships are correlated. When compared with other machine learning algorithm, Naive Bayes might not always emerge in term of of accuracy. However, its unparalleled efficiency and ease of deployment often tilt the scales in its favor, especially in scenarios demanding rapid data processing or real time classification. Figure 2.2 provides a comparative analysis of various classification algorithms, highlighting the strengths and weaknesses of each in relation to Naive Bayes.

2.3.2 Gaussian Naive Bayes

In this study, the supervised learning algorithm Naive Bayes Classifier will be investigate to be implemented on Red Pitaya to do the classification tasks . The Naive Bayes classifier is a probabilistic classifier that is based on Bayes' Theorem. The relationship between conditional probabilities of a hypothesis and observations, as shown in the equation 2.3.

$$P(H = C_k|O = o) = \frac{P(H = C_k)P(O = o|H = C_k)}{P(O = o)} \tag{2.3}$$

Denote T as a training set of samples, each with their class labels. There are k classes in the training set, $H = (C_1, C_2, ..., C_k)$. An n-dimensional vector, $O = (o_1, o_2, ..., o_n)$ represents n measured values of the n attributes,$(X_1, X_2, ..., X_n)$ respectively, each being the results of measurements of various feature O_i.

The Naive Bayes classifier assumes that the value of a particular feature is independent of the value of any other feature, given in same class. This assumption is called class conditional independence. The Naive Bayes classifier is based on the assumption that the features are independent given the class. Mathematically, the likelihood $P(O = o|H = C_k)$ can be stated as follows:

$$P(H = C_k|O = o) = \frac{P(H = C_k)\prod_{i=1}^{n} P(O_i = o_i|H = C_k)}{P(O_1 = o_1, ..., O_n = o_n)} \tag{2.4}$$

Since the denominator $P(O)$ is constant for all class label C_k, the conditional probability $P(H = C_k|O = o)$ can be written as:

$$P(H = C_k|O = o) \propto P(H = C_k)\prod_{i=1}^{n} P(O_i = o_i|H = C_k) \tag{2.5}$$

In Equation 2.6, these calculations are caried out on the log scale to prevent a numerical underflow (when $d \gg 0$)

$$\log P(H = C_k|O = o) \propto \log P(H = C_k) + \sum_{i=1}^{n} \log P(O_i = o_i|H = C_k) \tag{2.6}$$

Lastly, the class with the highest log-posterior probability is chosen to be the prediction:

$$\hat{C} = \arg \max_{k\in\{1,...,K\}} \left[\log P(H = C_k) + \sum_{i=1}^{n} \log P(O_i = o_i|H = C_k)\right] \tag{2.7}$$

The likelihoods of the features that has continuous attributes are typicalled modeled by Gaussian distributions in Naive Bayes classification. Therefore, a Gaussian probability density function(PDF) is utilized to define each attribute as $X_i \sim N(\mu, \sigma^2)$. The Gaussian PDF in Figure 2.3 has the shape of a bell and is defined by the following equation:

$$N(\mu, \sigma^2)(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.8}$$

where,

- μ is the mean,
- σ^2 is the variance,

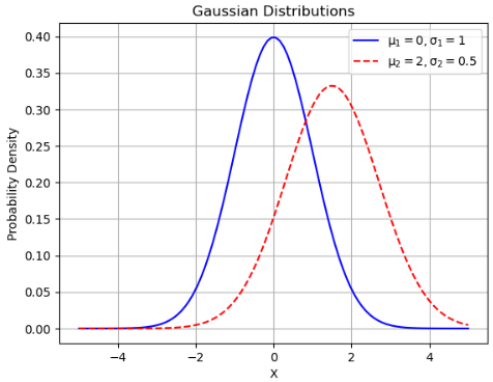


Figure 2.3: Gaussian distribution

3

HARDWARE SYSTEM

In this chapter, we will discuss the hardware setup required to develop a robust object classification system. Additionally, we will outline a methodology for creating a button interface that allows the Red Pitaya to manage both learning and prediction processes without the need for an external computer.

Ultrasonic waves are produced using high-frequency ultrasonic sensors. These sensors operate in conjunction with the Red Pitaya (RP) board, which is responsible for sending commands to the sensor and processing the data it collects. All components are interconnected via a custom circuit board *Michalik_RPUSv1.1* as depicted in Figure 3.1. This integrated setup facilitates the efficient generating, receiving, and processing of ultrasonic waves, enabling effective object detection and classification.

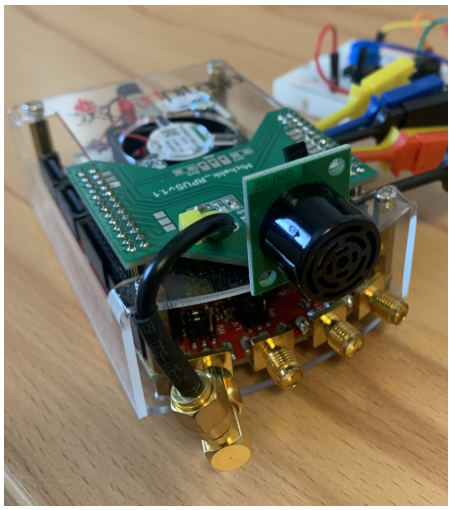


Figure 3.1: Ultrasonic sensor on Red Pitaya

3.1 Red Pitaya

The STEMLab 125-14 from RP had been selected as the hardware platform in this study. This single board computer serves as a cost-effective alternative to expensive laboratory measurement equipment. It offers a varieties functionalities, including an oscilloscope, signal generator, and spectrum analyzer, which are currently available in the repository [10].

The RP integrates a dual-core ARM Cortex A9 processor (866MHz) and Xilinx Zynq 7010 field programmable gate array (FPGA), making it well-suited for programming high-performance applications[9]. The implementation of complex machine learning classifier algorithms and sophisticated data acquisition benefits greatly from this feature. One of the main factors for choosing

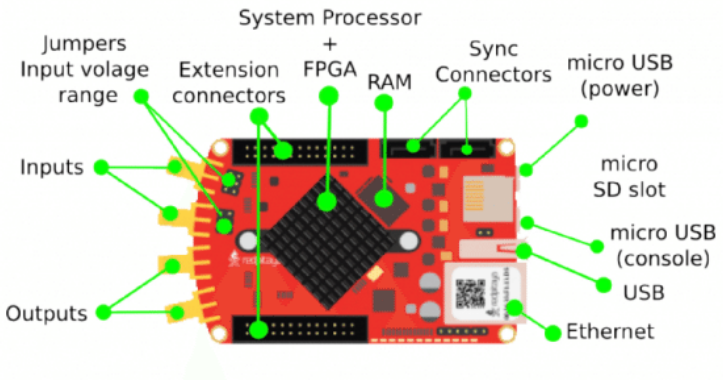
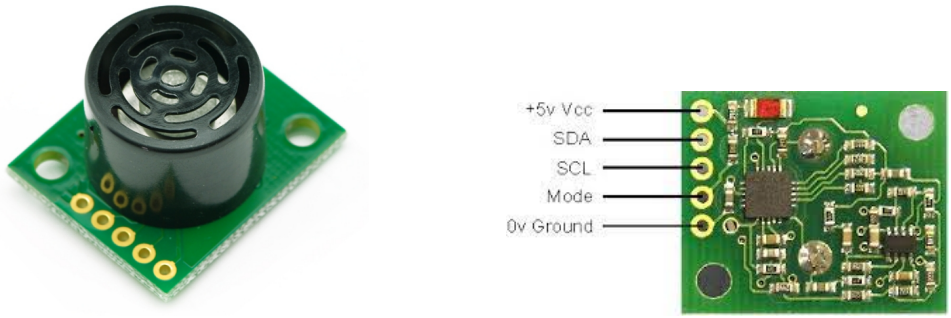


Figure 3.2: RedPitaya hardware overview [9]

the RP is its fast onboard analog input, which comes with a sampling rate of 125Msps. This high sampling rate is ideal for the research being conducted. Moreover, the Linux operating system's network stack, in conjunction with the Red Pitaya's ethernet port, makes it possible to integrate remote control functionality and facilitates the transfer of measurement data over the network.

3.2 SRF02 Ultrasonic Sensor



(a) Hardware Overview (b) I2C Communication

Figure 3.3: SRF02 Ultrasonic sensor [11]

To discern the characteristics of an object and estimate the distance between the sensor and an obstacle, ultrasonic sensors are typically composed of two modules: one that transmits ultrasonic waves and another that receives the reflected waves. For this project, the SRF02 ultrasonic sensor was chosen due to its integrated design and single probe that is capable of both transmitting and receiving waves. The reflected wave will come back to the sensor transducer if there is an object in the path of ultrasonic waves. This technique is called pulse echo method, as shown in figure 3.4 [12]. Unlike through transmission, in pulse echo method, only reflected signals are measureed. This method is simple as only one surface access is needed and it can be applied where it is

not possible of through transmission. Then, this signal is filtered, processed, and converted into digital form. A photograph of the SRF02 sensor can be seen in Figure 3.3a.

From the technical document [11], SRF02 operates at 5V, same as the power supply of the RP board, and draws only 5mA during operation, thereby making it a power-efficient sensor. With a measured distance range between 15 mm and 6 m, the SRF02 meets the requirements of the design's specifications. SRF02 offers versatility with two interface options: RS232 and I2C. For the I2C communication, SDA and SCL signals are connected to VCC by a 10K pull-up resistor, as shown in 3.3b.

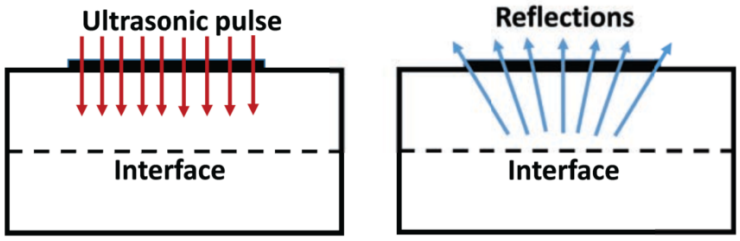
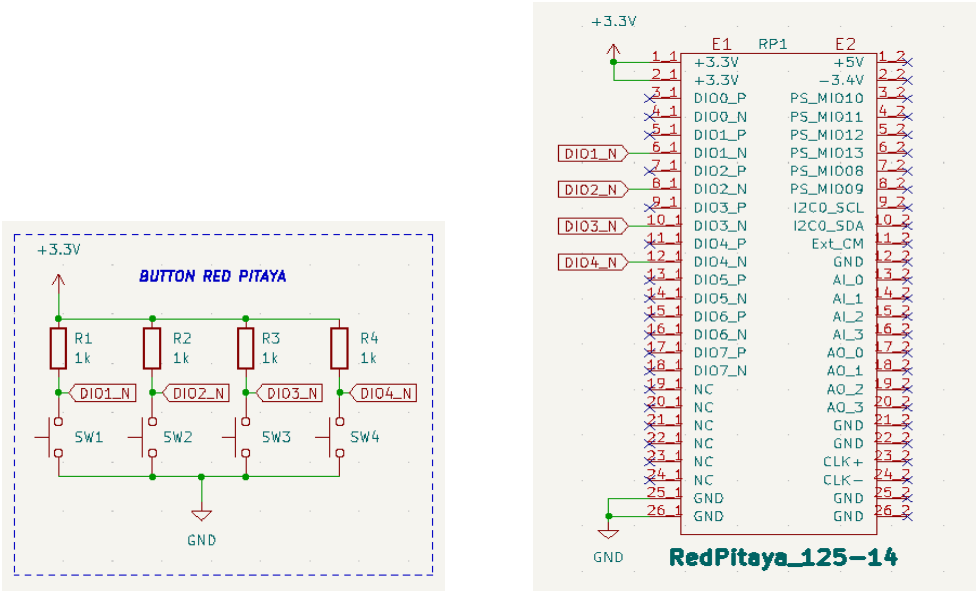


Figure 3.4: Pulse Echo Method

3.3 Button Interface



(a) On button

(b) On Red Pitaya

Figure 3.5: Button connection schematic

Four buttons have been connected to the RP board, utilizing the spare General Purpose Input Output (GPIO) for controlling the data processing of the

ultrasonic system. As shown in Figure 3.5, Buttons 1 through 4 are linked to pins D101 to D104, respectively, operating in pull-up mode.

Pull-up mode is a method used to define the state of a digital logic circuit[13]. In this configuration, an undefined signal is directed towards a high level through a resistor. A 1kΩ resistor was utilized according to the documentation to limit the current directed to the GPIO[9]. This ensures that the GPIO pins are operating within their safe current limits, preventing potential damage to the system.

In Figure 3.6, Each button has a unique function in the system's current state:

- Button 1: Logs non-human sensor data and saves it as a binary file.
- Button 2: Responsible for logging human data.
- Button 3: Runnning Box Classifier.
- Button 4: Running Naive Bayes Classifier.

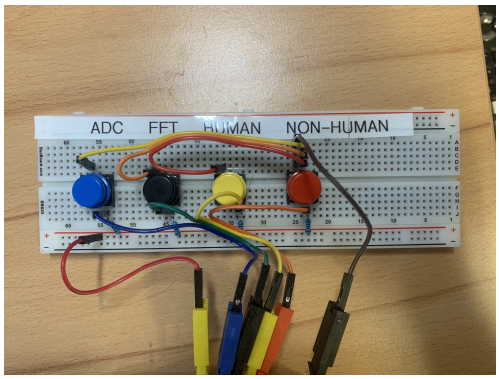
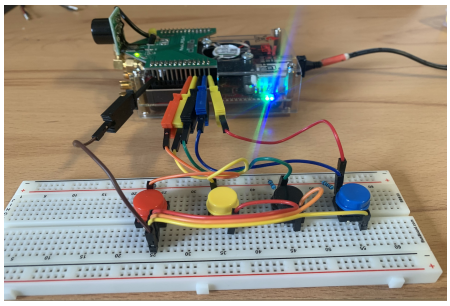
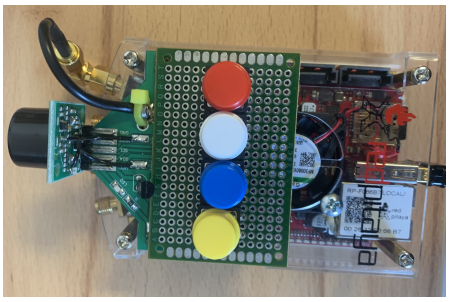


Figure 3.6: Buttons Function



(a) on Breadboard



(b) On PCB

Figure 3.7: Prototype of button interface

As illustrated in Figure 3.7a, the button interface is currently connected to the system using a breadboard to demonstrate proof-of-concept in this prototype phase. However, for long-term usage and increased reliability, a future enhancement to the design would involve switching from the breadboard to a

soldered PCB. This PCB will be mounted directly on top of the current system, as depicted in Figure 3.7b, producing a more durable and efficient setup for further research.

In learning phase, for saving feature data, our system employs a binary file format. The function responsible for this operation is:

Listing 3.1: Function to save sensor data to a binary file

```
int save_data_feature(void* feat, uint16_t number, const char*
  directory);

```

- **Parameters**:
  - `feat`: A pointer to the 10 feature extracted data.
  - `number`: Represents the classification of the data, e.g., human or non-human.
  - `directory`: Specifies the directory where the data will be stored , as can be seen in Figure 3.2.
- **Functionality**: This function is trigerred when users want to store or learn current feature set. It writes the feature data to a binary file in the designated directory. Binary files are chosen for their efficiency in storing vast amounts of data compactly and for their fast read/write capabilities.

Listing 3.2: Logging dataset

```
  |-human
  | |-human_feat_000.bin
  |-lib
  | |-librpxx.so
  |-nonhuman
  | |-nonhuman_feat_001.bin
  | |-nonhuman_feat_000.bin
```

In prediction phase, our sensor system is designed with flexibility in mind, allowing users to switch between different processing modes. This feature is implemented using the following function:

Listing 3.3: Function to set the running mode of the sensor system

```
  void iic_set_run(run_t run_in, bool param);

```

- **Parameters**:
  - `run_in`: Determines the running mode of the sensor system. Possible values include `run_bayes` for the Naive Bayes classifier and `run_box` for the traditional box classifier.
  - `param`: A boolean indicating whether the chosen mode should be activated or deactivated.

- **Functionality**: This function allows users to toggle between different processing modes, offering adaptability in data analysis and interpretation.

The system uses a set of onboard LEDs to provide visual feedback based on the model's predictions and the current operational mode. The behavior of these LEDs is depicted in Figure 3.8 below.

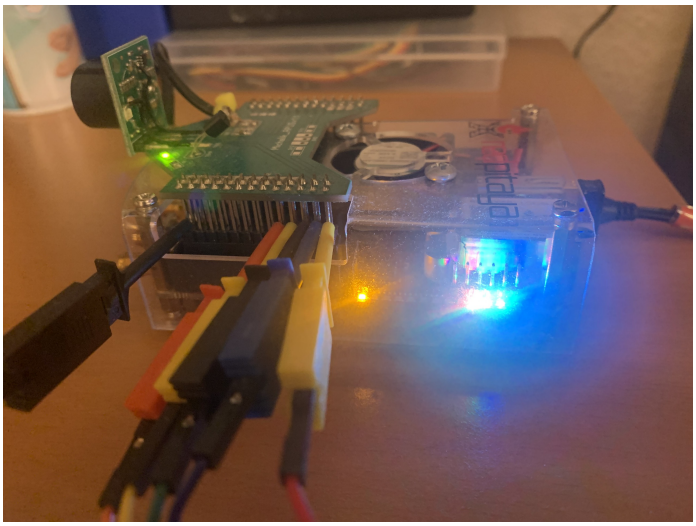


Figure 3.8: LED indication

- **LED 1 (Class 0 Indicator)**: Lights up when the model classifies the detected object as a soft object, typically indicating a human presence/ soft object.
- **LED 2 (Class 1 Indicator)**: Activates when the model classifies the detected object as a hard object, typically indicating a non-human entity/ hard object.
- **LED 3 (Classifier Mode Indicator)**: This LED is specifically linked to the operational mode of the model. It turns on when the system is utilizing the Naive Bayes Classifier. On the other hand, it remains off when the Box Classifier is in operation.
- **LED 6 (Learning Phase Indicator)**: This LED serves as an indicator of the system's learning phase. It lights up when the system is actively learning and storing feature data in binary files.

4

SOFTWARE IMPLEMENTATION

4.1 Ultrasonic Signal Processing

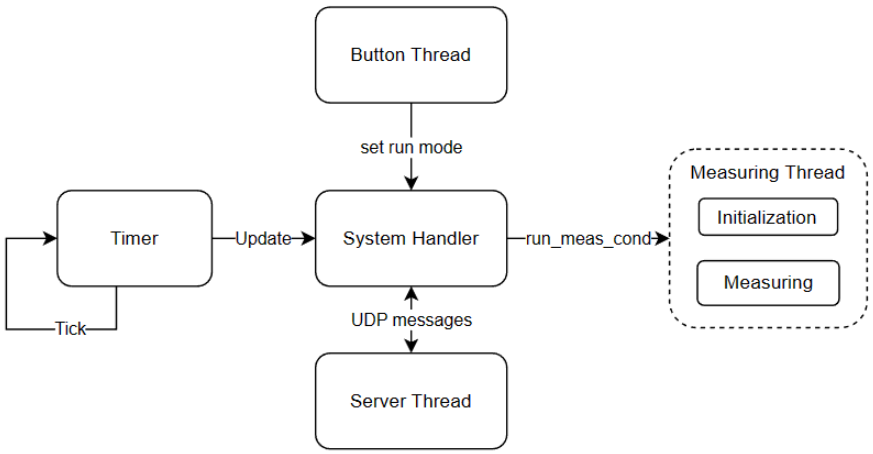


Figure 4.1: Software Architecture of the system

The software architecture on Red Pitaya is designed to be modular [14], as demonstrated in Figure 4.1 , with each thread performing a specific task.

- The Server Thread runs in the background, listening to User Datagram Protocol (UDP) messages at port 61231 in blocked mode. It parses the received commands and values.
- The Measure Thread initializes the Analog to Digital Converter (ADC) and Fast Fourier Transform (FFT) processes and operates in blocked mode under control of Time Thread
- The Time Thread manages the measuring process, which takes place every 100ms.
- The Button Thread provides a Human Machine Interface (HMI) to control the sensor board. It allows the board to function as a standalone device without the need for a computer.

Figure 4.13 illustrates multiple sequential steps of the operating process of our system. First, the ultrasonic sensor is activated to gather object data, which is then preprocessed using the *kiss_fft* library to convert into FFT data [15].

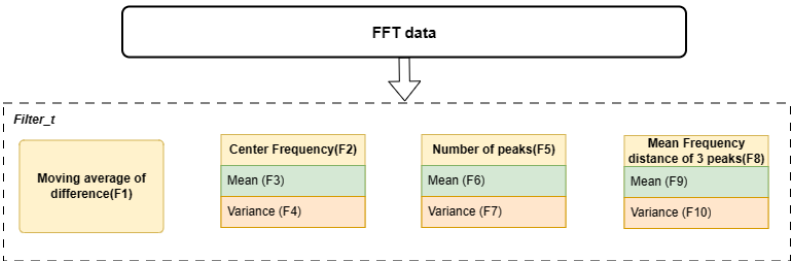


Figure 4.2: Overview of 10 Features

Subsequently, the data undergoes a filtration process, wherein specific attributes are extracted to discern the object's unique characteristics. 10 Features have been identified, as shown in Figure 4.2. The mean and variance derived from the primary Filter (F2, F5, F8) are passed through additional features, respectively.

Feature 1: The mean difference between the frequencies of adjacent measurements of the center frequency is calculated. This feature provides insights into how the center frequency is changing over time or space. A higher mean difference suggests a more dynamic center frequency, suggest greater variability or fluctuations in the center frequency, while a lower mean difference might indicate more stability.

Feature 2 (Center Frequency): The center frequency represents the predominant frequency at which the energy of the signal is most concentrated. This feature is particularly important when assessing the frequency response of materials or mediums using ultrasonic sensors. For instance, in ultrasonic testing, the center frequency can provide insights into the resonant frequency of a material, aiding in its characterization. However, a limitation to consider is that while it excels at identifying the dominant frequency, it might not fully capture the complexity of ultrasonic signals with multiple dominant frequencies.

Feature 4 (Numbers of Peaks): This feature involves counting the number of peaks present in the frequency domain representation of the ultrasonic signal. By doing so, it provides information about the complexity and the number of dominant frequencies in the signal. It's particularly useful in applications where understanding the frequency composition of the signal is essential. While it provides a comprehensive overview of the signal's complexity, discerning genuine peaks from those induced by extraneous noise may necessitate further analytical processes.

Feature 8 (Mean 3 Peak Frequency Distance): By measuring the average distance between the three most prominent peaks (left of center frequency, center frequency, right of center frequency) in the frequency domain, Feature 8 offers insights into the distribution of the main frequency components. A larger mean distance suggests a wide dispersion of these frequencies across the spectrum, while a smaller mean distance indicates a more clustered distribution. This metric is invaluable in scenarios where understanding a signal's frequency composition is paramount. However, it may not be suitable for signals with multiple dominant frequencies. Furthermore, same as Feature 4, the

presence of noise or non-dominant frequencies can skew the accuracy of the feature, necessitating rigorous preprocessing or filtering to ensure the analyzed peaks genuinely represent the signal's dominant frequencies.

After these analytical processes, the data is either parsed into a classifier for classification or archived as a binary file for future utilization.

4.2 GUI Implementation

4.2.1 Overview

The evolution of our sensor data acquisition and processing software has been significant. The initial version, V0.22, was primarily focused on basic functionalities. This Graphical User Interface (GUI) can log received data and save it in a text file or visualizes data sourced from the system. This software version offered the following functionalities as depicted in Figure 4.3.

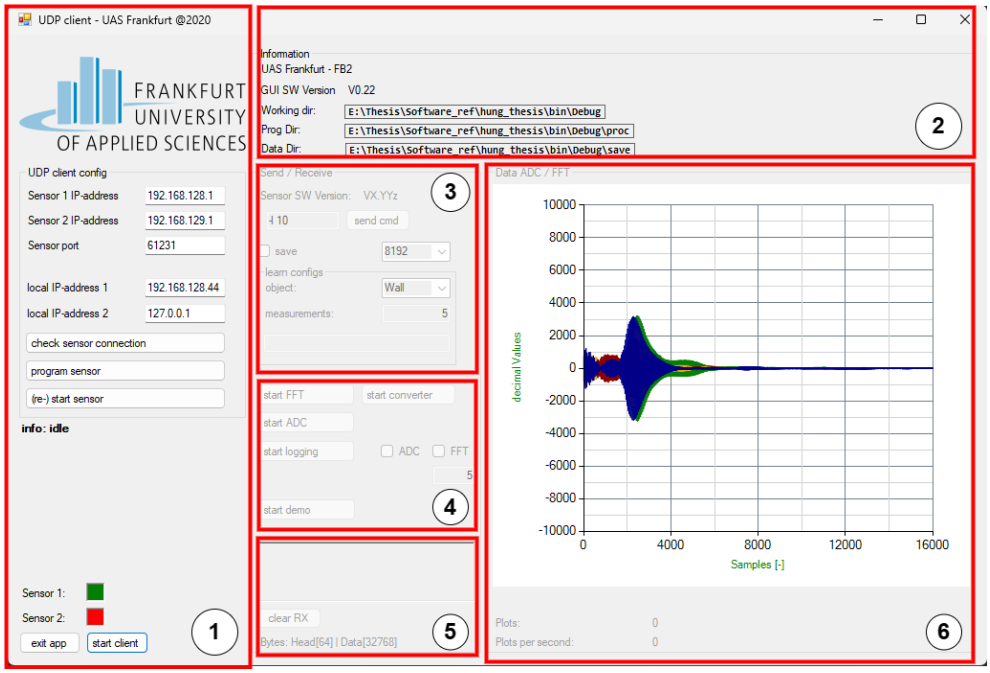


Figure 4.3: Fra-UAS UDP Client

1. UDP Parameter Settings includes the sensors' addresses and UDP ports configuration.

2. Directory Paths displays the directories for the GUI, source code, and recorded data.

3. Command Prompt box sends UDP commands to the sensor module.

4. Operation Controls allow starting the measurement process in "FFT" or "ADC" mode or switching to logging mode.

5. Received Data Box showcase incoming features from sensor,such as distance and object class.

6. Data Visualization box provides a graphical representation of the "FFT" or "ADC" signals.

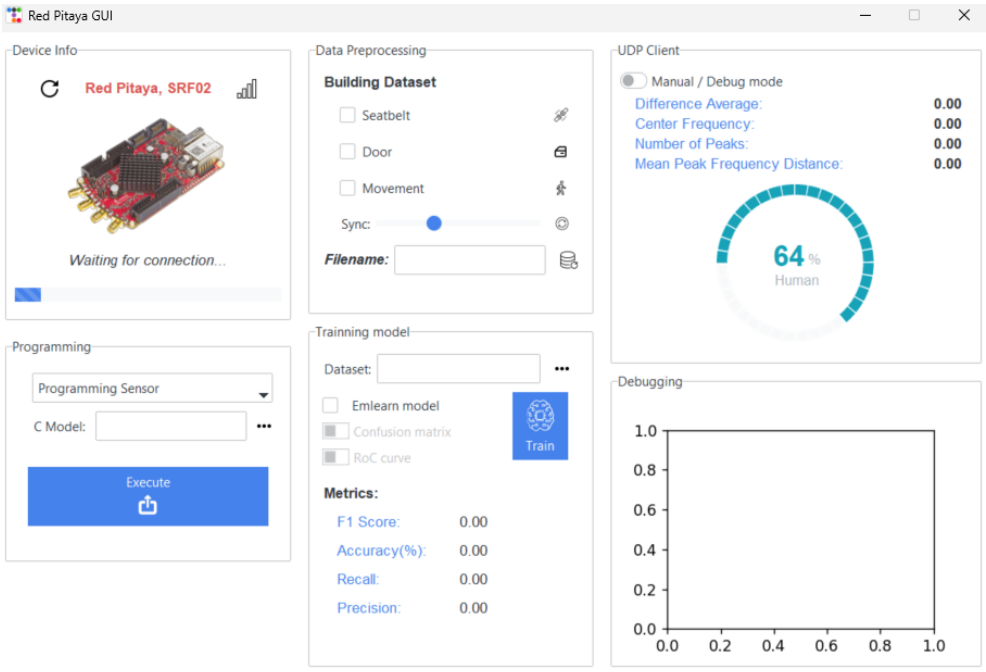


Figure 4.4: Smart Sensor Application

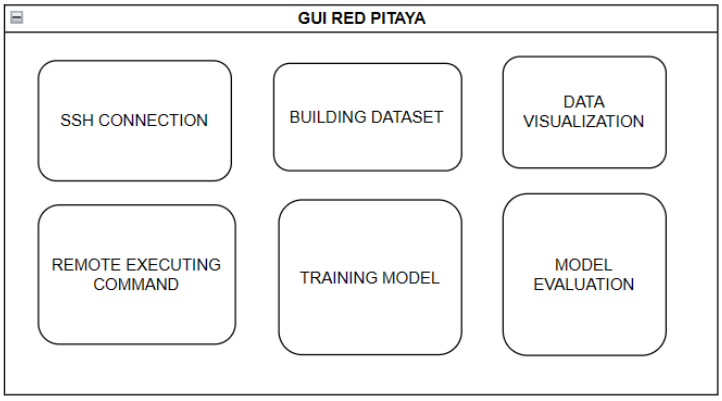


Figure 4.5: Functionalities of Smart Sensor Application

However, recognizing the lack of functionalities, particularly in training and integrating of machine learning model, we introduced the new Smart Sensor Application 4.5. This new version marks a significant advancement. A screenshort of the application can be seen in 4.4 .It not only allows users to construct datasets based on ultrasonic signals but also integrates a robust

Machine Learning section for training and evaluating new models. One of its standout features is the ability to remotely deploy new models to the sensor. This, combined with the capability to compile new firmware, simplifies operation steps and pipeline, making it more user-friendly and efficient. Furthermore, the Feature Data Visualization offers a more intuitive monitoring experience. Instead of solely relying on traditional ADC or FFT graphs, users can now view extracted features, which can be crucial for in-depth analysis and system diagnostics. Developed in Python, this application is supported by a wide range of machine learning and signal processing frameworks, paving the way for future research. The function of each section in the application will be discussed in more detail below.

4.2.2 Remote Command Execution

The smart sensor system's GUI is designed to execute predefined remote command with the help of 'paramiko' library [16]. This Python Library provides an interface for SSH communication, allowing users to send commands directly to the sensor system from a remote computer. Those commands are executed in real time, and the feedback on the sensor status and performance are displayed on the GUI. The remote command execution section is shown in Figure 4.6. For ease of use and to ensure efficient operation of the sensor system, a

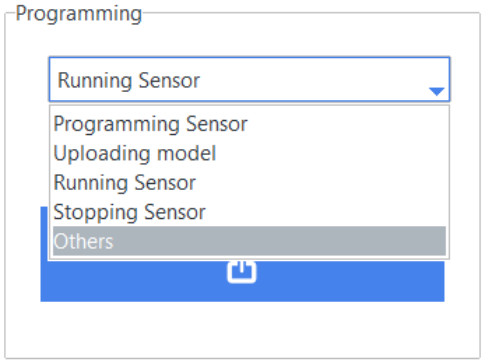


Figure 4.6: Remote Command Section

set of predefined commands has been established. These commands are listed in 4.1 below:

Users can utilize these commands to perform specific operations on the Red Pitaya platform. For instance, the `make` command is used for building the new firmware after uploading the new machine learning model, while the `$(./sensor_start.sh)` command initiates the sensor's operation with linked library. To halt the sensor, the `kill -9 $(pgrep iic)` command can be executed to stop the process.

Table 4.1: Commands for Various Operations on Red Pitaya

Option	Command
Building firmware	`make`
Running sensor	`./sensor_start.sh`
Stopping sensor	`kill -9 $(pgrep iic)`

4.2.3 Building Dataset

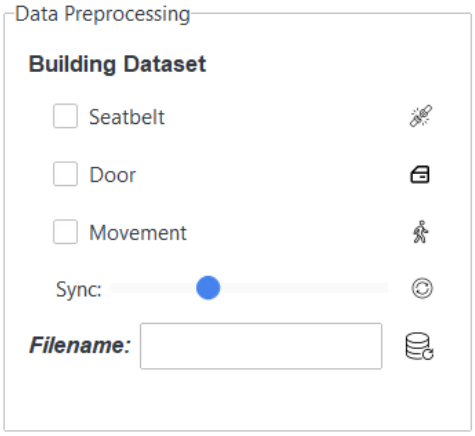


Figure 4.7: Building dataset

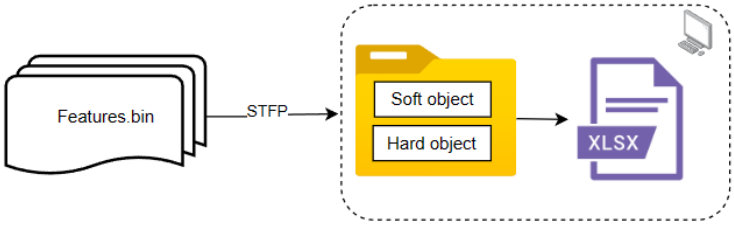


Figure 4.8: Operating steps of building dataset

After the learning phase, our features are stored on the Red Pitaya device. The primary goal before constructing a dataset for model training is to synchronize these saved files back to the local computer. This synchronization is achieved using the Secure File Transfer protocol(SFTP). SFTP is a robust and secure method for transferring file over TCP/IP newwork. As a subsystem of SSH, it encompasses all SSH protocols, including those for authentication and encryption. The implementation of the SFTP protocol is realized through the Paramiko library, which is integrated into our GUI, as illustrated in Figure 4.7.

The dataset filenames are named based on specific experimental conditions. The primary considerations include:

- **Seatbelt Status**: Indicated as either 'on' or 'off'.
- **Door Status**: Denoted as 'open' or 'closed'.
- **Movement Detection**: Presence or absence of movement.

Each file is embedded with a timestamp for storage and future research purposes. Once these binary files are retrieved, they undergo a conversion process to be transformed into '.xlsx' format, as showcased in 4.8. The dataset contains 10 essential features, with each data entry being appropriately labeled. A label of '1' denotes human object, while '0' indicates non-human object. An representative example of our dataset is presented in Table 4.2.

Table 4.2: Example of dataset

Feature 1	Feature 2	Feature 3	Feature 6	Label
0.00	41007.00	41007.00	2.90	1
11.90	40888.00	40995.10	3.10	1
11.90	40888.00	40983.20	3.20	1
11.90	40888.00	40971.30	3.00	1
11.90	40888.00	40959.40	3.10	1
11.90	41127.00	41234.10	1.10	0
23.80	41246.00	41234.10	1.20	0
35.70	41127.00	41222.20	1.30	0
47.60	41246.00	41222.20	1.30	0
47.60	41246.00	41222.20	1.40	0

4.2.4 Training Model and Implementing on Red Pitaya

In the progression of our research, following the data preparation detailed in section 4.2.3, we moved on the pivotal phase of our project: the model construction. The Gaussian Naive Bayes (GNB) algorithm was chosen by its suitability for our dataset's characteristics and the nature of the problem at hand.

The Python-based machine learning library, Scikit-learn (often abbreviated as `sklearn`) [17], was employed to facilitate the construction of our GNB model. This library offers a comprehensive suite of tools for data mining and data analysis, making it an apt choice for our project. To train our model, we utilized the `GaussianNB` class from the `sklearn.naive_bayes` module. The typical workflow involved initializing the model with `gnb = GaussianNB()`, followed by calling the `fit` method with our dataset to train the model: `gnb.fit(X_train, y_train)`. Here, `X_train` represents the feature matrix, and `y_train` denotes the corresponding labels. This process allowed the algorithm to learn the underlying patterns and relationships present in our data.

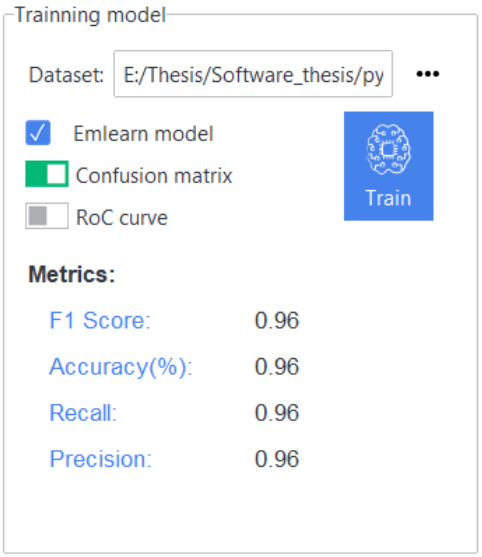


Figure 4.9: Training data

Post-training, the next challenge was to deploy the model on the Red Pitaya, an embedded system. Given the constraints and specificities of embedded systems, it was imperative to convert our Python-based model into a format amenable to such an environment. This is where `Emlearn` came into play.

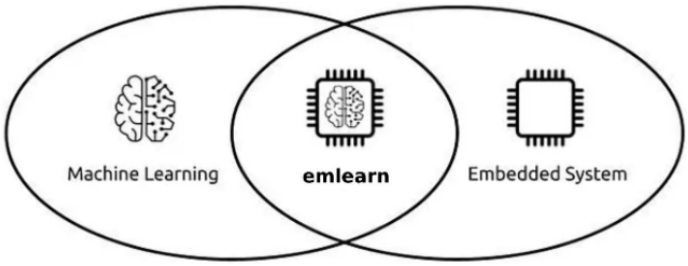


Figure 4.10: Overview of Emlearn library

The Emlearn library represents a state-of-the-art Tiny Machine Learning Framework, tailor-made for embedded systems [18]. It converts Python to C model, serving as an inference engine, and is compatible with most of popular microcontrollers and other devices operating on the C99 programming language. Emlearn supports a diverse range of machine learning models, including neural networks (NN), random forests (RF), decision trees (DT), naive Bayes (NB), multilayer perceptrons (MLP), and sequential models constructed using the Keras and scikit-learn frameworks. A standout feature of Emlearn is its avoidance of dynamic memory allocation, opting instead for fixed-point math. This choice enhances efficiency, ensuring stability and predictability in embedded environments. Notably, Emlearn is also compatible with diffrent single board computer such as Raspberry Pi 4, as demonstrated by Gustavo de Carvalho Bertoli in 2021 [19]. Given its unique feature set, Emlearn emerges as an ideal choice for this study, which seeks to integrate machine learning capabilities into embedded platforms.

By using this library, our GNB model was transformed into a C header file and uploaded to our system with the help of uploading command , as mentioned in section 4.2.2. The conversion process resutled the following C array, `EmlBayesSummary model_f10_summaries[8]`, which encapsulates the summaries for each feature, notably the mean, standard deviation, and the natural logarithm of the standard deviation for each feature:

Listing 4.1: Model Summaries

```
EmlBayesSummary model_f10_summaries[8] = {
  { EML_Q16_FROMFLOAT(61.549061719109034),EML_Q16_FROMFLOAT
  (64.59159938897986),EML_Q16_FROMFLOAT(6.013274638940599) },
  % ... [other data points]
  { EML_Q16_FROMFLOAT(1.2079900122165084),EML_Q16_FROMFLOAT
  (0.29136904281348563),EML_Q16_FROMFLOAT(-1.779080491856858) }
};
```

In the code snippet above, the `EML_Q16_FROMFLOAT` macro converts floating-point numbers into a fixed-point format. In digital systems, real numbers are primarily represented using two methods: fixed-point representation and floating-point representation [20]. While both have their advantages, their efficiency varies significantly when deployed on microcontrollers. The mathematical expression for a fixed-point number is given by:

$$x = i + f \times 2^{-n}$$

where:

- i is the integer component of the number.
- f denotes the fractional part.
- n is the fixed number of bits allocated for the fractional component.

Suppose we use a fixed-point format with 16 bits in total, where 10 bits are allocated for the integer part and 6 bits for the fractional part. The number 6.4123 can be approximated and represented as 6.40625 in binary, which translates to 110.01101. In a 16-bit format, this would be stored as 0000001100110100. On the other hand, in a simplified floating-point format, numbers are represented using a sign bit, an exponent, and a fraction. According to the IEEE 754 standard for single-precision floating-point numbers (32 bits), 6.4123 would be represented with a specific sign bit, an 8-bit exponent, and a 23-bit fraction [21]. The preceise binary representation would be more complex due to the normalization process and bias added to the exponent.

On microcontrollers, especially those without a dedicated floating-point unit (FPU), fixed-point arithmetic offers several advantages over floating-point arithmetic:

1. **Computational Speed:** Fixed-point arithmetic operations are straightforward, making them faster on microcontrollers without a dedicated floating-point unit (FPU). In contrast, floating-point operations involve manipulating the sign, exponent, and fraction, which can be computationally intensive.

2. **Memory Efficiency:** Fixed-point numbers use a consistent number of bits, leading to predictable memory usage. In contrast, floating-point numbers can vary in size, leading to potential memory inefficiencies. The fixed-point representation of 6.4123 uses a consistent 16 bits. However, the floating-point representation, even in our simplified example, would require 32 bits. This makes fixed-point more memory-efficient for this specific number

3. **Precision:** The fixed-point representation approximated 6.4123 as 6.40625, introducing a small error. Floating-point, with its dynamic range and larger bit-width for the fraction, can represent numbers with higher precision, but it's also prone to rounding errors, especially for very large or very small numbers.

Algorithm 4: Emlearn Gaussian Naive Bayes on Red Pitaya

Input : EmlBayesSummary(mean, std, stdlog2), Input values.
Output : Predicted class: either 0 (non-human) or 1 (human)
Function `EmlearnNaiveBayes()`:

1) Convert all input values to fixed point representation;
2) Calculate ln of the PDF using a quadratic approximation;
3) Compute ln of $N(\mu, \sigma^2)$ based on the standard $N(0, 1)$;
4) Utilize the function `eml_bayes_predict` to:

- Compute the log-posterior probabilities for each class
- Identify the class with the highest log-posterior probability

return Predicted class;

Listing 4.2: PDF of standard normal distribution

```
1 eml_bayes_logpdf_std(eml_q16_t x)
```

As illustrated in Algorithm 4, the implementation of a Gaussian Naive Bayes model on Red Pitaya involves several steps. Initially, all parameters are converted to fixed-point representation. Subsequently, the PDF of the standard normal distribution is calculated, serving as a reference for the subsequent calculation of P(feature|label). The function 4.2 computes the natural logarithm of the probability density function (PDF) of a standard normal distribution with a mean of 0 and a standard deviation of 1 for a given value of x in fixed-point representation.. The function is implemented using a quadratic approximation of the PDF.

The equation for the logarithm of the standard Gaussian PDF is:

$$\ln(f(x)) = -\frac{1}{2}\ln(2\pi) - \frac{x^2}{2} \tag{4.1}$$

To approximate this equation, the function adopts a polynomial of the form:

$$y = ax^2 + bx + c \tag{4.2}$$

Where:

$$a = -0.7213475204444817$$
$$b = -1.0005845727355313 \times 10^{-15}$$
$$c = -1.3257480647361592$$

These coefficients were likely determined to ensure that the polynomial closely aligns with the logarithm of the Gaussian PDF within a specific range of x values. An error analysis of this approximation is provided below.

Given:

- Polynomial Approximation: $f(x)_{\text{approx}} = e^y$ where y is the output of the `eml_bayes_logpdf_std` function.
- Direct Computation: $f(x)_{\text{exact}} = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

Percentage Error formula:

$$\text{Percentage Error} = \left| \frac{f(x)_{\text{approx}} - f(x)_{\text{exact}}}{f(x)_{\text{exact}}} \right| \times 100\%$$

Using the provided formulas, let's compute the values and percentage errors for each x close to zero:

x	$f(x)_{\text{approx}}$	$f(x)_{\text{exact}}$	Percentage Error
0.1	$\approx e^{-1.3979}$	$\approx \frac{1}{\sqrt{2\pi}} e^{-0.005}$	24.6%
0.01	$\approx e^{-1.3265}$	$\approx \frac{1}{\sqrt{2\pi}} e^{-0.00005}$	2.46%
0.001	$\approx e^{-1.3257}$	$\approx \frac{1}{\sqrt{2\pi}} e^{-5\times10^{-7}}$	0.246%
0.0001	$\approx e^{-1.3257}$	$\approx \frac{1}{\sqrt{2\pi}} e^{-5\times10^{-9}}$	0.0246%

Table 4.3: Comparison of Polynomial Approximation and Exact Gaussian PDF

As x approaches zero, the accuracy of the polynomial approximation improves, with the error decreasing significantly. This indicates that the polynomial approximation is optimized for values of x close to zero.

The transformation from any Gaussian distribution to the standard form is achieved using the z-score, defined as:

$$z = \frac{x - \mu}{\sigma} \tag{4.3}$$

To express the PDF of a Gaussian distribution in terms of the standard normal distribution in logarithmic form, one can derive the following relationship:

$$\ln(f(x|\mu, \sigma^2)) = \ln(f(z)) - \ln(\sigma) \tag{4.4}$$

where $f(z)$ is the PDF of the standard normal distribution which was calculated using quadratic approximation. The logarithm of the standard deviation,

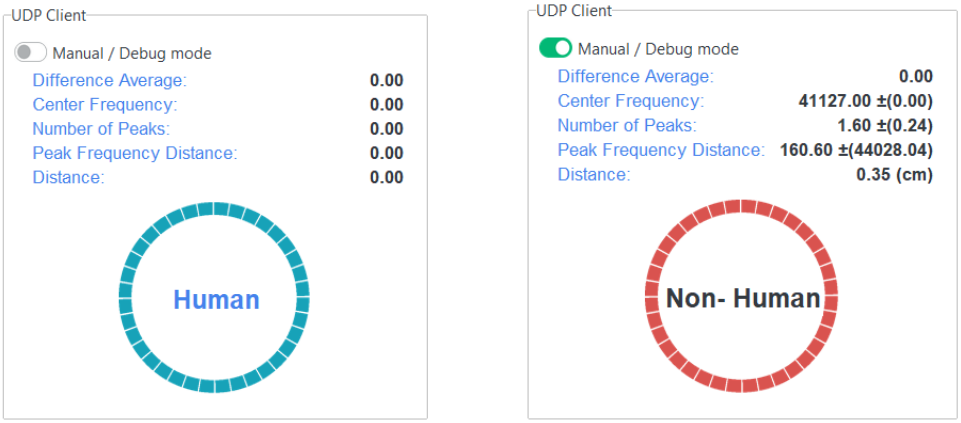
$\ln(\sigma)$, is precomputed and stored in the `stdlog2` field of the `EmlBayesSummary` structure.

The final step involves calculating the log-posterior probabilities for each class and identifying the class with the highest log-posterior probability. This is accomplished using the `eml_bayes_predict` function, as highlighted below:

Listing 4.3: Classification

```
1 int32_t eml_bayes_predict(EmlBayesModel *model, const float
      values[], int32_t values_length);
```

4.2.5 Data Visualization



(a) Soft Object (b) Hard Object

Figure 4.11: Data Visualisation

Our application is also focused on real-time data visualization, aiming for effective debugging and in-depth analysis. The feature data, captured by the sensor, is transmitted to the GUI using the UDP packet. This protocol was chosen for its efficiency in delivering real-time data without the need for a continuous connection.

Upon arrival at the GUI, the data's structure is visually represented, as seen in Figure 4.11a for human-centric data and Figure 4.11b for non-human data. The GUI presents this data in a structured table format. Some of the pivotal features highlighted are the Difference Average, Center Frequency, Number of Peaks, and the Mean Peak Frequency Distance, each paired by their respective mean and variance values.This Feature representation supports in the understanding and interpretation of the object properties.

As the sensor continually captures and sends new data packets, this UDP clients updates in real-time. This ensures that researchers always have access to the most recent data. Furthermore, the GUI's interactive visualization, as depicted in Figure 4.11, offers users the flexibility to delve deeper into the data, enabling thorough research and analysis.

4.2.6 Operating steps

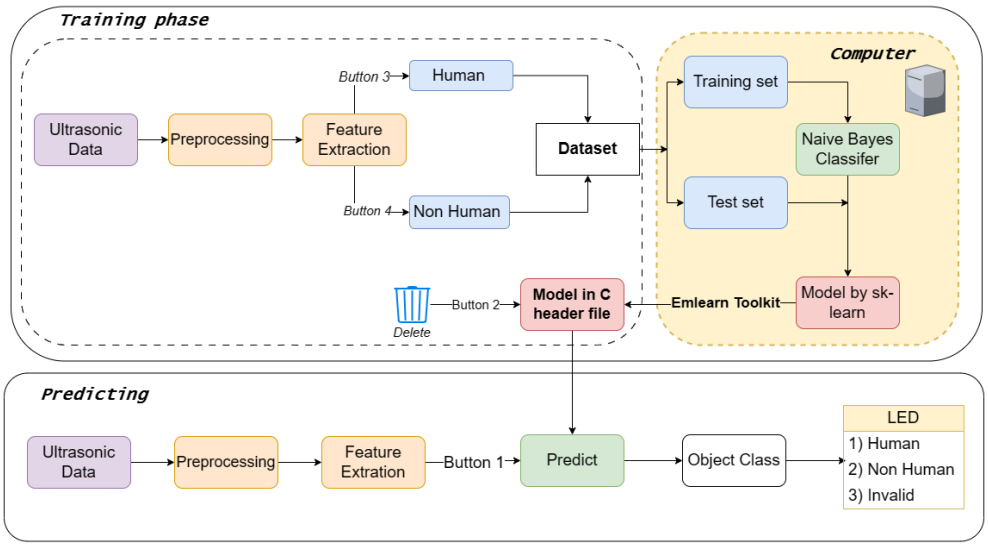


Figure 4.12: Machine Learning Pipeline Overview

The Machine Learning Pipeline, as depicted in Figure 4.12, is structured into two primary phases: the training phase and the prediction phase.

Training Phase

During the training phase, feature data is gathered from both human subjects and non-human objects using the ultrasonic sensor. The construction of a labeled dataset is facilitated by Buttons 3 and 4. Specifically, pressing Button 3 triggers the system to store feature data derived from human subjects, whereas Button 4 serves a similar function for non-human objects. The accumulated features are preserved in binary files. Subsequently, these files are merged, transformed into a cohesive dataset, and transferred to a computer for training. Once processed, the dataset becomes the foundation for training the Gaussian Naive Bayes Classifier model.

To ensure a robust evaluation of the model's performance, the dataset is bifurcated into two subsets: 80% of the data is splitted for training, while the remaining 20% is reserved for testing. This division is a standard procedure in Machine Learning, enabling the evaluation of the model's performance on previously unseen data. Following the model's training via sklearn, the Emlearn toolkit is employed to convert the model into a C header file, facilitating its integration into a C application and subsequent deployment on the Red Pitaya platform utilizing the `scp` command, as detailed in 4.2.2. The procedural steps for uploading a fresh model to the sensor are delineated in Figure 4.13. After compilation and linkage new C model with the sensor program, the sensor stands ready for prediction tasks.

Prediction Phase

In the prediction phase, the act of pressing Button 1 activates the prediction function. The system then switchs to classifying mode, basing its decision on the maximal posterior probability as computed by the Naive Bayes Classifier.

The classification outcome is visually displayed through one of the LEDs (LED 1, 2, or 3), with each LED corresponding to a distinct object class. Further analysis of the prediction results can be using the GUI debugging section 4.2.5.

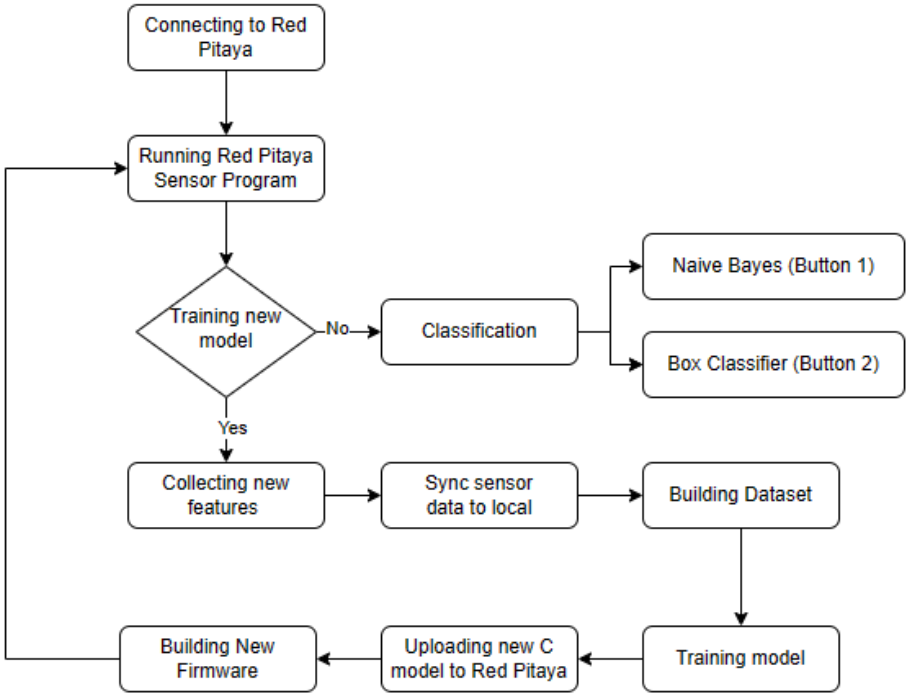


Figure 4.13: Operating steps of Smart Sensor System

4.3 Cronjob

In earlier projects, the sensor's startup was traditionally tied to UDP client software operated from a laptop. This setup resulted in the sensor system being dependent on the software. To enhance our process automation and transform our sensor into a standalone device, we incorporated a Unix-based job scheduler known as 'Cron' into our system design.

Cron is a powerful tool in Unix operating systems for scheduling tasks, which can be commands or scripts that need to run at specific times or on specific days [22]. This utility offers substantial flexibility in system management, enabling precise task automation, which in turn improves system performance.

We created a script file to automate the startup of our sensor system in order to take advantage of Cron's capabilities. The script was written in Bash, a scripting language used in Unix computers. The script is as follow:

Listing 4.4: sensor start script

```
1 #!/bin/bash
2 cd /pathtosensor
3 LD_LIBRARY_PATH=/opt/redpitaya/lib ./iic
```

The first line of the script indicates to the system that this script should be executed as a Bash script. Following this, the daemon is instructed to navigate to the sensor program directory using the `cd` command. The script then uses the `./iic` command to initiate the sensor, ensuring that it links with the Red Pitaya board's API by adding the `LD_LIBRARY_PATH` tag.

Our specific Cron command, created to execute this script, is as follows:

Listing 4.5: cronjob

```
1 @reboot sleep 20 && sh /root/redpitaya/Examples/Communication/C
    /iic/proc/sensor_start.sh >dev/null 2>&1
```

The string `'>dev/null 2>&1'` appended at the end of the command serves to discard all output and standard error messages, ensuring a clean console and preventing unnecessary message logging by redirecting both output and error messages to the null device.

Through the implementation of Cron, the sensor system can now automatically start at bootup, our dependency on manual client operations has been reduced, which improved our overall system efficiency and flexibility.

5.1 Experimental setup

5.1.1 Experimental cases

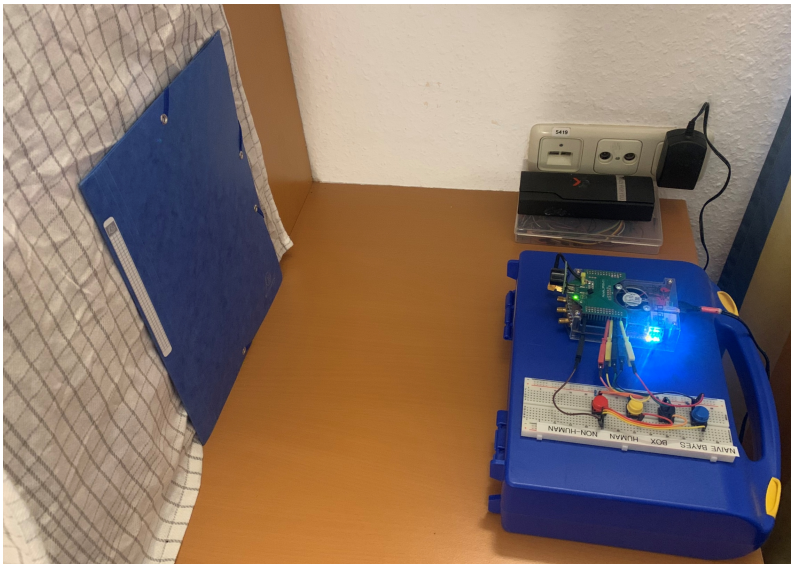


Figure 5.1: Experimental Setup

Figure 5.1 illustrates the experimental setup for our sensor system. The sensor is positioned on a stable surface. To prevent the transducer's operating blind zone and to obtain a more accurate and effective ultrasonic echo signal, the sensor system is placed 35 cm directly away from the material being measured.

In this experiment, we selected paperboard and a blanket (representing a soft object) as the target materials to differentiate. The dimensions of the boards for all three materials are 50 cm $\times$ 50 cm, each with a thickness of 1 mm. Measurements were repeated 10 times, and an average was taken to eliminate data noise. We ensured that an equal amount of data was collected for each material to prevent dataset imbalance.

For each case, varying data sample sizes were collected: 200 samples, 2,000 samples, and 20,000 samples. This variation aims to provide insights into how sample size impacts our model's performance. Once the dataset was compiled, we performed various metric evaluations on it and analyzed the results.

5.1.2 Model Evaluation

In order to thoroughly evaluate the performance of the proposed model, we employed several widely recognized metrics, namely accuracy, sensitivity, specificity, precision, and F1-score. These metrics are defined using the variables TP (true positives), TN (true negatives), FP (false positives), and FN (false negatives). The model collected data throughout the training, validation, and testing phases, and the metrics were computed using the aforementioned features. Each metric provides insights into different aspects of the model's performance.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{5.1}$$

Accuracy offers a general overview of the model's correctness, but it may not be sufficient for applications where the data is imbalanced, such as in autonomous systems. In the context of ultrasonic signal classification based on object properties, where each object's features are unique, it is crucial for the model to accurately identify outliers. Sensitivity and specificity provide insights into the model's ability to correctly classify samples within a class and identify negatives, respectively.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{5.2}$$

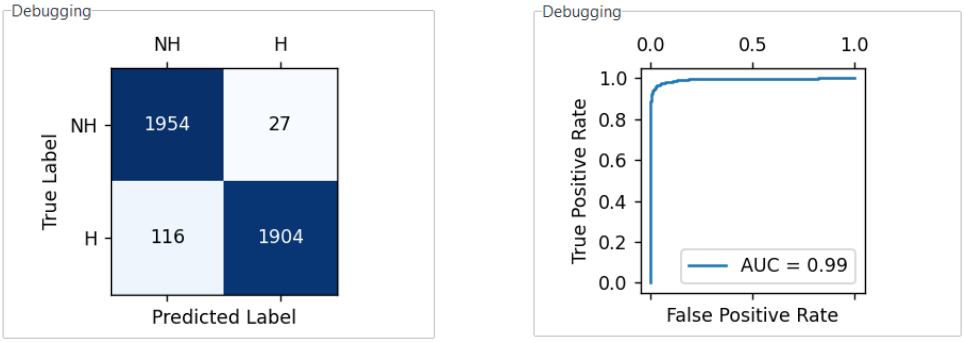
$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{5.3}$$

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

Precision is particularly valuable for imbalanced datasets as it indicates the proportion of correct classifications for each class. By combining sensitivity and precision, the F1-score offers a holistic view of the model's performance for each class.

To provide a comprehensive evaluation of the model, it is also important to discuss the confusion matrix and AUC (Area Under the Curve). The confusion matrix provides detailed information about the model's performance across different classes. The AUC score is used to evaluate the performance of a binary classification model. A higher AUC indicates a better model performance.

Our sensor application also supports model evaluation, as illustrated in Figure 5.2. In this context, the metrics described above are essential for ensuring the reliability and accuracy of the model, especially when dealing with real-world data and scenarios. Each metric serves a unique purpose in highlighting the strengths and potential areas of improvement for the model.



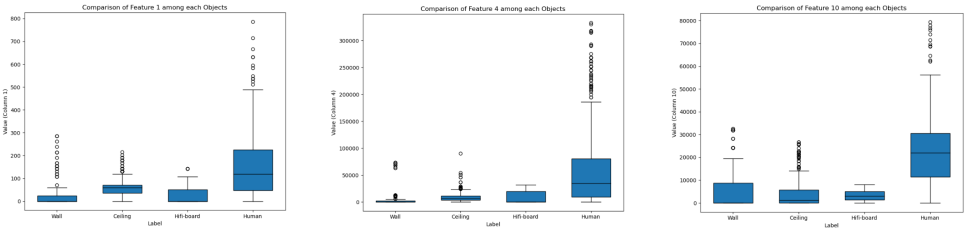
(a) Confusion Matrix (b) AUC Graph

Figure 5.2: Machine Learning model evaluation on Smart Sensor Application

5.2 Experimental Results

5.2.1 Feature Selection

When performing classification, one of the most crucial decisions to be made is to choose appropriate features. Previous research often used Features 1, 4, and 10 to determine the box classifier threshold. Therefore, we examine these in Figure 5.3a, a box plot comparing Feature 1 across different objects. Boxplots show data distribution based on five-number categories: minimum, first quartile ($Q1$), median, third quartile ($Q3$), and maximum. Feature 1, the only feature using the moving average of the difference filter, provides separate values for different classes, with visible difference between human and non-human objects. In addition, features using the moving variance filter produce large data values, usually creating clearer class boundaries, as shown in Figures 5.3. However, in Figures 5.3b and 5.3c, a large number of outliners was noticed. We believe that these outliners are caused by the uncetainties of the sensor during the measurement process. Hence, we will need to carefully select features for the dataset to improve the model's accuracy.



(a) Feature 1 (b) Feature 4 (c) Feature 10

Figure 5.3: Boxplots of Features

Boxplot shapes can also indicate if a dataset is normally distributed or skewed. While most figures show a skewed distribution, the human object boxplot in Figure 5.3c exhibits equal proportions around the median, creating a

Gaussian distribution. This feature is crucial when calculating likelihood probabilities with Gaussian Naive Bayes. A normal distribution simplifies equations and reduces complexity; otherwise, discretization method is necessary to convert continuous data into categorical data.

Independence between features is a pivotal consideration when employing Naive Bayes classifiers. The correlation map of the 10 features from class 0 is depicted in Figure 5.4. Some features, specifically 3 and 4 from feature 2, 6 and 7 from feature 5, and 9 and 10 from feature 8, exhibit correlation due to their derivation from a shared feature. To uphold model accuracy, all features that are positively correlated should be discarded. As a result, features such as 1, 3, 6, and 10 have been chosen for the dataset.

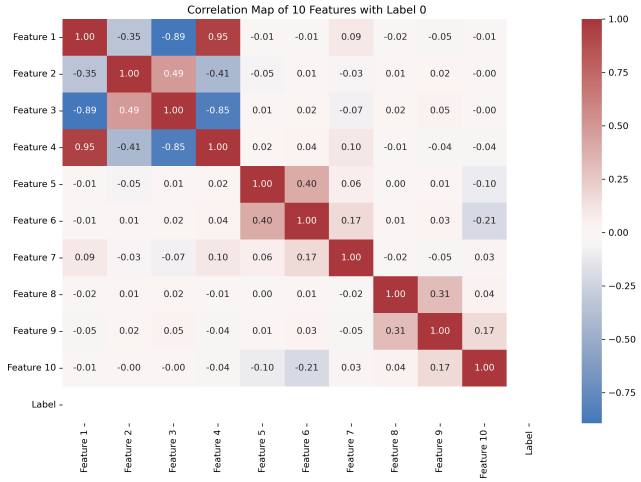


Figure 5.4: Correlation map of label 0

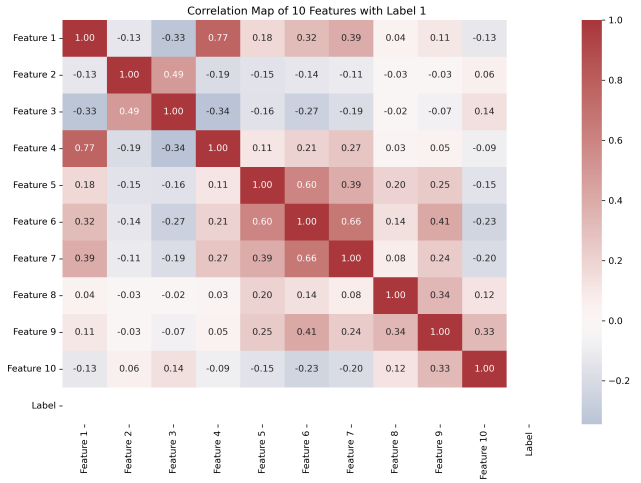


Figure 5.5: Correlation map of label 1

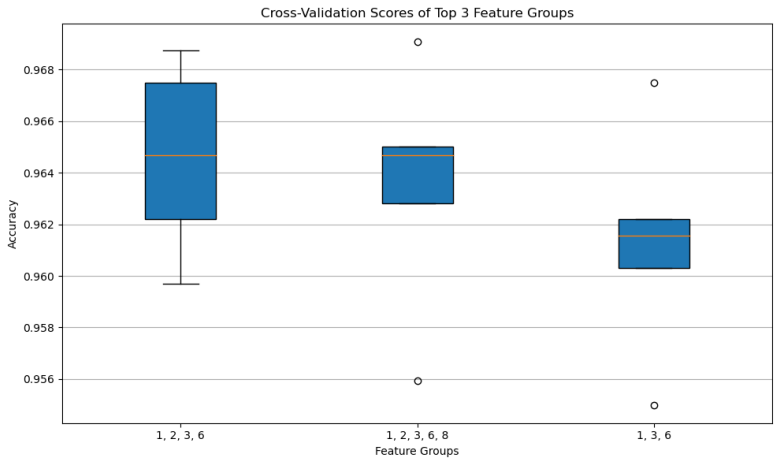
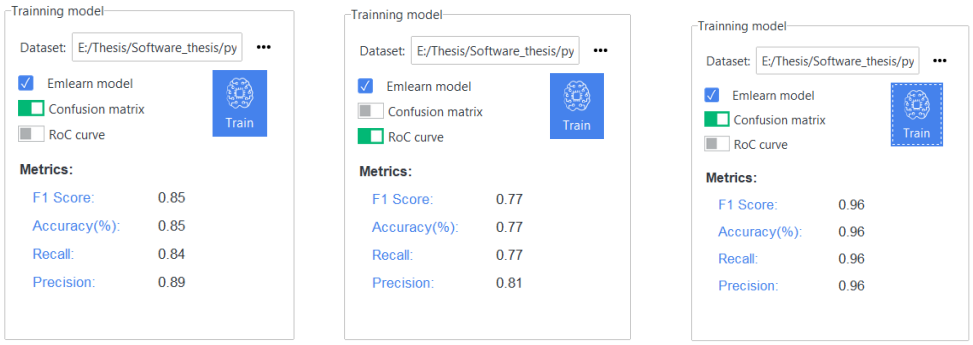


Figure 5.6: Cross Validation scores of top 3 features

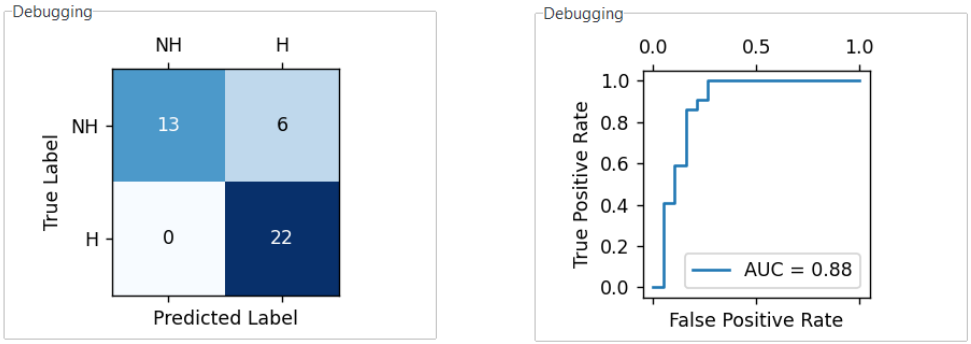
The cross-validation score was computed using the sklearn tool. As illustrated in Figure 5.6, feature groups 1, 3, and 6 demonstrated superior accuracy. These features were suitable for the Gaussian Naive Classifier because they are uncorrelated and follow to a normal distribution. The next phase of the reserach will focus on evaluating the accuracy of the real-world acoustic model based on this feature selection group.

5.2.2 Machine Learning Model Performance



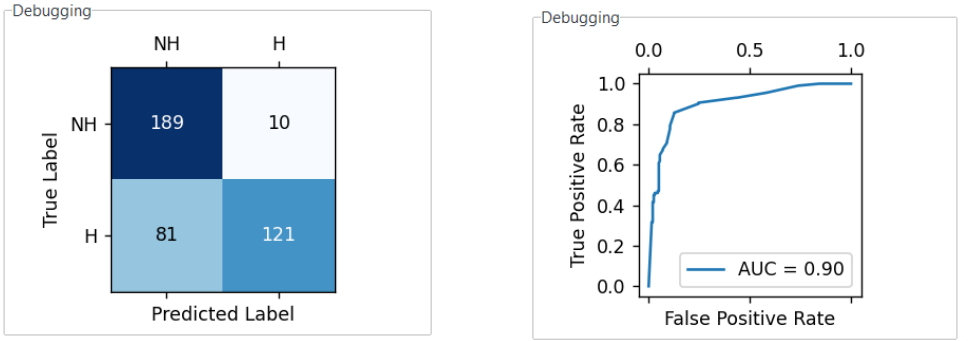
(a) 200 samples (b) 2000 samples (c) 20000 samples

Figure 5.7: Model evaluation with different sample size



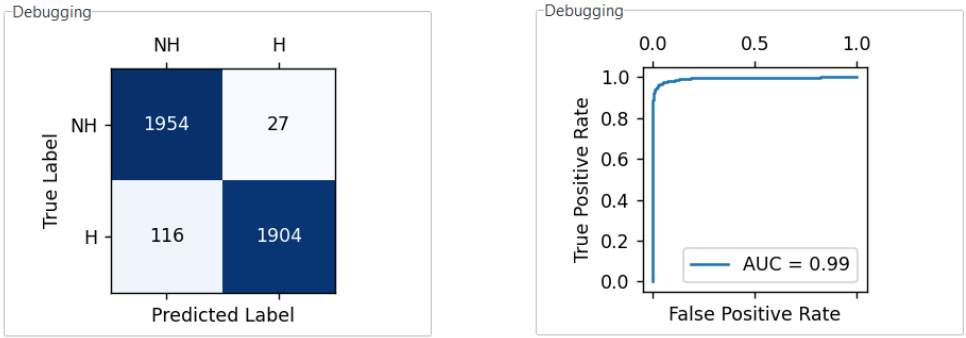
(a) Confusion Matrix (b) AUC Graph

Figure 5.8: Confusion Matrix with 200 samples



(a) Confusion Matrix (b) AUC Graph

Figure 5.9: Confusion matrix with 2,000 samples



(a) Confusion Matrix (b) AUC Graph

Figure 5.10: Confusion matrix with 20,000 samples

Table 5.1: Classification Results between Wooden wall and Blanket

Method	Number of Samples		
	200	2,000	20,000
Sklearn	85%	77%	96%
Real-world	90%	88%	82%

The confusion matrix for various classification algorithms is depicted in Figure 5.10. From this figure, it's showcased that the recognition accuracy between blanket and sheet cover is notably reliable, registering at approximately 85%, even with a sample dataset size of just 200.

We integrated the previously discussed acoustic theoretical model for our analysis. Using the equations detailed in an earlier section, we computed the real-world model evaluation metrics for different materials. These results are showcased in Table 5.1. A comparison of the classification results between the real-world and the sklearn model reveals that with 20,000 samples, there appears to be overfitting. The sklearn model boasts a 96% accuracy, but this drops to 82% in real-world testing. Intriguingly, among all sample sizes, the 200-sample dataset delivers the most balanced classification results. Consequently, we've identified it as the optimal data size.

5.3 Discussions

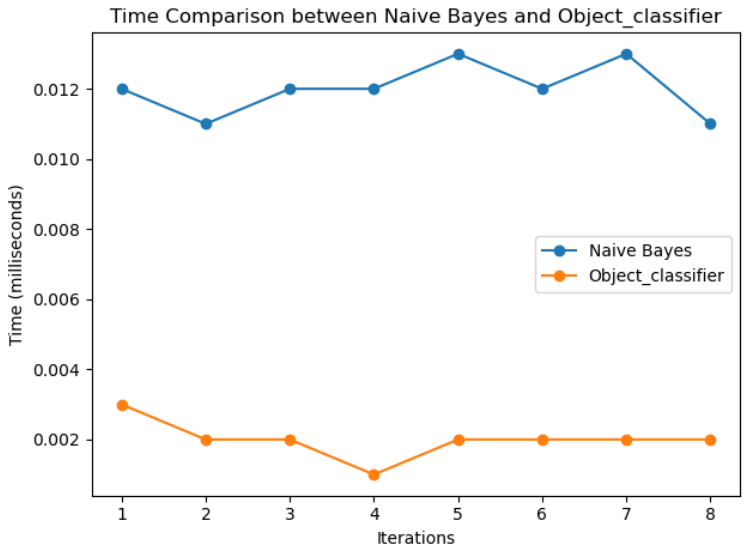


Figure 5.11: Comparison of execution time between Emlearn and Box classifier

By utilizing the `time.h` library, we have successfully measured the execution time of two distinct machine learning algorithms. Figure 5.11 showcases a comparative analysis of the execution time between the Emlearn Naive Bayes classifier and a traditional box classifier. In general, the box classifier exhibits superior efficiency, registering an execution time of approximately 0.004 mil-

liseconds, in contrast to the 0.012 seconds recorded by the Naive Bayes classifier. In terms of CPU consumption, the traditional methodology further shows its advantage, using a mere 13 microseconds, as opposed to the slightly more demanding 16 microseconds required by the Emlearn model, as illustrated in Figure 5.12.

However, a important metric to consider is accuracy. The Emlearn GNB demonstrates superior performance in this regard. When tasked with differentiating between a wooden wall and a soft object, such as a blanket, it showcases an accuracy rate of approximately 91% over a testing duration of 5 minutes, underscoring its reliability. On the otherhand, the box classifier necessitates sophisticated calibrations to find an optimal threshold value and, even then, achieves a relatively modest accuracy rate of around 75%, as showcased by other previous works.

In summation, while the Emlearn model might exhibit a marginally higher power consumption, its superior accuracy makes it a first choice, especially in applications where precision in object classification is paramount.

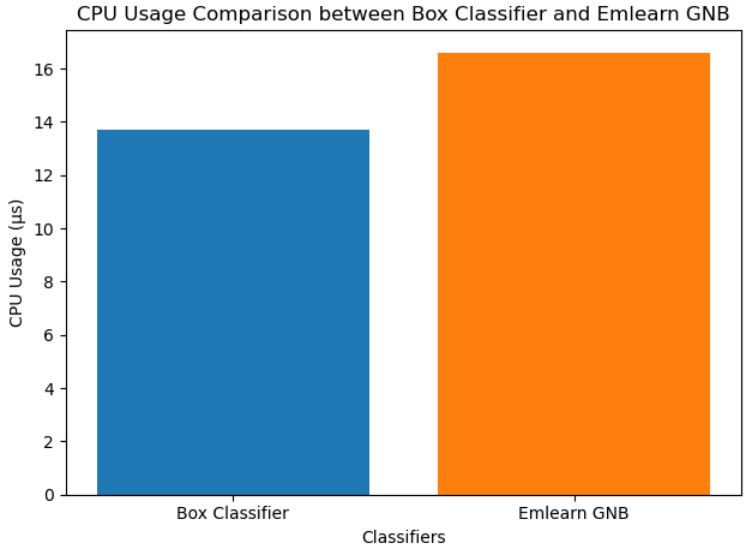


Figure 5.12: Comparison of CPU Usage between Emlearn and Box classifier

5.4 Limitations and Future Work

Ultrasonic sensor signals, when integrated with the Naive Bayes classifier for object classification, still show certain limitations that can hinder their efficiency. To begin with, the current classifier is binary, meaing it is designed to differentiate between only two classes simultaneously. This binary classifier has its limitation, especially in multifaceted scenarios. For instance, in a real-world scenario, there might be a need to differentiate between a human and diverse objects such as a wooden wall, a blanket, or a metallic surface. In such cases, the binary classifier would be unable to provide the necessary

insights. To address this limitation, future research should focus on developing a multi-class classifier that can distinguish between a wide range of objects.

Another area of concern is the processing technique that employs the Fourier Transform. While it provides frequency content, it does not offer insights into how this frequency content evolves over time. This time-based information is crucial, particularly when the objective is to differentiate between materials with inconsistent or dynamic textures. Some material properties can not be extracted with current feature extraction process. Given the importance of this temporal aspect, future research should delve into the application of techniques like the short-term Fourier Transform and wavelet transform.

Lastly, another limitation of the current model is its incapacity to identify and categorize unfamiliar objects. The model's prediction relies on the comparsion of the posterior probabilities from input feature, leading it to classify objects based on the class with the predominant probability. Consequently, when confronted with an unknown object, the model invariably associates it with the class bearing the higher probability. This approach is problematic as the model remains unreliable to the distinction between recognized and unrecognized objects. It is essential for upcoming research to concentrate on formulating a model equipped to categorize unfamiliar objects as well.

6

CONCLUSION

In this research, our goal was to explore the potential for classifying different materials as either soft or hard based on raw ultrasonic data. We successfully built a HMI using button inputs to record object data features, which were stored as binary files. Using Crontab, we ensured the system's automatic startup at bootup, making it a truly standalone, intelligent sensor system independent of external software for initialization or control. We conducted a thorough analysis of our data features to evaluate their suitability for implementing a Naive Bayes Classifier. Additionally, we utilized the Emlearn machine learning framework to implement a model on Red Pitaya. Experimental results indicate that the Naive Bayes Classifier achieved 90% accuracy and performed well even with a small dataset. These results also validate the accuracy of the acoustic theoretical model and the effectiveness of our proposed method. Compared to previous work, our method enhances the accuracy and recognition effects in ultrasonic material recognition.

While the outcomes of this project are promising, there are areas for future exploration. The application needs further refinement in terms of error handling, and the feature extraction process warrants additional study to identify more valuable features. Overall, research on object differentiation using ultrasonic sensors running on Red Pitaya contributes to a range of applications, from robotics to autonomous systems, inspiring further advancements in the fields of Embedded Systems and Machine Learning.

BIBLIOGRAPHY

[1] Donghee Yi, Heetae Jin, Moon Chan Kim, and Suk Chan Kim. "An Ultrasonic Object Detection Applying the ID Based on Spread Spectrum Technique for a Vehicle." In: *Sensors* 20.2 (Jan. 11, 2020), p. 414. ISSN: 1424-8220. DOI: 10.3390/s20020414. URL: https://www.mdpi.com/1424-8220/20/2/414.

[2] G Arun Francis, M Arulselvan, P Elangkumaran, S Keerthivarman, and J Vijaya Kumar. "Object Detection Using Ultrasonic Sensor." In: *Int. J. Innov. Technol. Explor. Eng* 8 (2020), pp. 207–209.

[3] Bo Zhu, Tao Geng, Gedong Jiang, Zheng Guan, Yang Li, and Xialun Yun. "Surrounding Object Material Detection and Identification Method for Robots Based on Ultrasonic Echo Signals." In: *Applied Bionics and Biomechanics* 2023 (May 15, 2023), e1998218. ISSN: 1176-2322. DOI: 10.1155/2023/1998218. URL: https://www.hindawi.com/journals/abb/2023/1998218/.

[4] Mohammad Rubaiyat Tanvir Hossai, Md. Asif Shahjalal, and Nowroz Farhan Nuri. "Design of an IoT Based Autonomous Vehicle with the Aid of Computer Vision." In: *2017 International Conference on Electrical, Computer and Communication Engineering (ECCE)*. 2017 International Conference on Electrical, Computer and Communication Engineering (ECCE). Feb. 2017, pp. 752–756. DOI: 10.1109/ECACE.2017.7913003. URL: https://ieeexplore.ieee.org/abstract/document/7913003?casa_token=ityNpZ6EQqgAAAAA:5a_rxtfx4g3GsM3Wd95zuCUKOZBzw56CZ1an1H1rd_8BmXJ1A44KVwpDZgSDrcl1J_YEU4zDLVve6w.

[5] Taiwo Oladipupo Ayodele. "Machine learning overview." In: *New Advances in Machine Learning* 2 (2010), pp. 9–18.

[6] FY Osisanwo, JET Akinsola, O Awodele, JO Hinmikaiye, O Olakanmi, J Akinjobi, et al. "Supervised machine learning algorithms: classification and comparison." In: *International Journal of Computer Trends and Technology (IJCTT)* 48.3 (2017), pp. 128–138.

[7] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen. "Naïve Bayes." In: *Encyclopedia of machine learning* 15.1 (2010), pp. 713–714.

[8] SL Ting, WH Ip, Albert HC Tsang, et al. "Is Naive Bayes a good classifier for document classification." In: *International Journal of Software Engineering and Its Applications* 5.3 (2011), pp. 37–46.

[9] *Red Pitaya Documentation*. URL: https://redpitaya.readthedocs.io/en/latest/.

[10] *RedPitaya Repository*. URL: https://github.com/RedPitaya/RedPitaya.

[11] *SRF02 Documentation*. URL: http://robot-electronics.co.uk/htm/srf02tech.htm.

[12] M.S. Beck. *Process Tomography: Principles, Techniques and Applications*. Elsevier Science, 2012. ISBN: 978-0-08-093801-1. URL: https://books.google.de/books?id=ebCGAAAAQBAJ.

[13] *Pull-up Resistor*. URL: https://learn.sparkfun.com/tutorials/pull-up-resistors/all.

[14] Robert La Lau. "Task Scheduling." In: *Practical Internet Server Configuration*. Berkeley, CA: Apress, 2021, pp. 175–181. ISBN: 978-1-4842-6959-6 978-1-4842-6960-2. DOI: 10.1007/978-1-4842-6960-2_8. URL: https://link.springer.com/10.1007/978-1-4842-6960-2_8.

[15] Mark Borgerding and Mark@Borgerding.net. *A FFT Library That Tries to Keep It Simple*. URL: https://github.com/mborgerding/kissfft.

[16] *Welcome to Paramiko! — Paramiko Documentation*. URL: https://www.paramiko.org/.

[17] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project." In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[18] Jon Nordby. *Emlearn: Machine Learning Inference Engine for Microcontrollers and Embedded Devices*. Mar. 2019. DOI: 10.5281/zenodo.2589394. URL: https://doi.org/10.5281/zenodo.2589394.

[19] Gustavo De Carvalho Bertoli, Lourenco Alves Pereira Junior, Osamu Saotome, Aldri L. Dos Santos, Filipe Alves Neto Verri, Cesar Augusto Cavalheiro Marcondes, Sidnei Barbieri, Moises S. Rodrigues, and Jose M. Parente De Oliveira. "An End-to-End Framework for Machine Learning-Based Network Intrusion Detection System." In: *IEEE Access* 9 (2021), pp. 106790–106805. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3101188. URL: https://ieeexplore.ieee.org/document/9501960/.

[20] Pierre-Emmanuel Novac, Ghouthi Boukli Hacene, Alain Pegatoquet, Benoît Miramond, and Vincent Gripon. "Quantization and Deployment of Deep Neural Networks on Microcontrollers." In: *Sensors* 21.9 (9 Jan. 2021), p. 2984. ISSN: 1424-8220. DOI: 10.3390/s21092984. URL: https://www.mdpi.com/1424-8220/21/9/2984.

[21] *IEEE Standard for Floating-Point Arithmetic*. IEEE. DOI: 10.1109/IEEESTD.2019.8766229. URL: https://ieeexplore.ieee.org/document/8766229/.

[22] Larry Reznick. "Using cron and crontab." In: *Sys Admin* 2.4 (1993), pp. 29–32.

A

APPENDIX

The enclosed archived file uploaded to the Frankfurt University of Applied Sciences contains the followings attachments:

1. **Source Code of the Application**
2. **Text of the Thesis in PDF Format**
3. **Video Demonstration of the Application:** Watch Video