STUDY AND EVALUATION OF BIOMETRIC TECHNIQUES: SPEAKER RECOGNITION

Anna Vittoria Damato

Antonio Nardone

Carolina Sbiroli

Raffaele Di Benedetto

Academic Year 2024/2025

Electronic systems for Biometrics and biosensing

Table Of Contents

1 OBJECTIVES OF THE REPORT 11
2 HARDWARE 12
2.1 Components 12
2.1.1 STM32 Nucleo F091RC 12
2.1.2 Sparkfun Electret Microphone Breakout – CMA544PF With Amplifier - MAX9814 14
2.2 Circuits 15
2.2.1 Microphone 15
3 FIRMWARE 16
3.1 MBED OS 16
3.1.1 Voice Acquisition 17
3.2 Matlab 18
3.2.1 Speaker Recognition 19
4 RESULTS 27
4.1 Speaker Recognition 27
5 CONCLUDING NOTES AND POSSIBLE OBSERVATIONS 38
5.1 Observations 38
5.2 Conclusion 39

Table Of Figures

Figure 1: Outline of the identification process in a biometric system 6
Figure 2: Outline of the authentication process in a biometric system 7
Figure 3 : Hamming Window 8
Figure 4: (a) Relationship between the linear frequency scale (Hz) and the Mel scale (mel); (b) Mel-Scale Filter Bank Composed Of Seven Filters 9
Figure 5: STM32 NUCLEO F091RC 12
Figure 6: Sparkfun Electret Microphone 14
Figure 7: Circuit Diagram For Speech Acquisition 15
Figure 8: Firmware For 16bit And 4000Hz Audio Sampling With Microphone 17
Figure 9: Code for Audio Signal Processing and User Recognition Using MFCC 24
Figure 10: Noise Signal Obtained By Averaging The Three Pure Noise Acquisition 25
Figure 11: Steps Of The Pre-Processing Phase 26
Figure 12: Microcontroller and Electret Microphone 27
Figure 13: Denoised Signals Truncation Through Two Thresholds For Surname "Damato" 29
Figure 14: Denoised Signals Truncation Through Two Thresholds For Surname "Di Benedetto" 30
Figure 15: Denoised Signals Truncation Through Two Thresholds For Surname "Nardone" 31
Figure 16: Principal Component Analysis And Centroids Definition (Enrollment Phase) 34
Figure 17: Principal Component Analysis And Centroids Definition (Identification Phase) 35
Figure 18: (a) Example Of Saturated Signal; (b) Focus On Saturation At 2450 mV 38

Table Of Tables

Table 1: Speaker Recognition Results For The Surnames 'Damato', 'Nardone' And 'Di Benedetto' 36
Table 2: Accuracy Score As The Normalization Technique Used Changes 37

Acronyms

STM	STMicroelectronics
LED	Light Emitting Diode
USB	Universal Serial Bus
COM	Communication
DAC	Digital-to-Analog Converter
ADC	Analog-to-Digital Converter
ST	STmicroelectronics
IDE	Integrated Development Environment
HAL	Hardware Abstraction Layer
SNR	Signal to Noise Ratio
FET	Field Effect Transistor
DC	Direct Current
Wi-Fi	Wireless Fidelity
PC	Personal Computer
FFT	Fast Fourier Transform
OS	Operating System
RISC	Reduced Instruction Set Computer
Arm	Advanced RISC Machines
MATLAB	MATrix LABoratory
RTOS	RealTime Operating System
IoT	Internet of Things
IPv6	Internet Protocol version 6
TLS	Transport Layer Security
DTLS	Datagram Transport Layer Security
MQTT	Message Queuing Telemetry Transport
CoAP	Constrained Application Protocol
FAT	File Allocation Table
LittleFS	Little Flash File System
CMSIS	Cortex Microcontroller Software Interface Standard
RTX	RealTime Ray Tracing
MFCC	Mel-Frequency Cepstral Coefficients
PCA	Principal Component Analysis

Definitions

I. **Biometrics:** the discipline that studies the measurement of the unique characteristics of the human being, both physical and behavioral.

II. **Biometric technologies:** the set of all techniques that reliably exploit measurable physiological or behavioral characteristics to identify and distinguish an individual from others.

III. **Biometric recognition:** a process that allows the identity of a person to be identified or verified based on unique physical or behavioral characteristics. The general process involves several phases, starting with the acquisition of the biometric data, which is then subjected to pre-processing to improve its quality. Next, distinctive characteristics are extracted that can be stored in a database (enrollment) for later comparison. When a new biometric sample is captured, it is compared to those already stored to determine the user's identity.

IV. **Speaker Recognition:** biometric recognition technique that is based on the analysis of voice samples from multiple users. It consists of acquiring a vocal sample during the recording phase and transformed into a digital representation. Subsequently, when the user wants to access a system, a new voice sample is acquired and compared with those stored through template extraction and comparison techniques.

V. **Identification Process:** a process that allows an individual to be recognized by comparing their biometric characteristics with those stored in a database. The process begins with the acquisition phase, during which biometric data is collected, which can be taken from images, audio or physiological signals. Subsequently, distinctive characteristics are extracted from this data, a fundamental step in obtaining a biometric template that can be compared with others already present in the database.

At this point, the "1-to-many" comparison takes place, in which the new sample is compared with all the biometric information stored to identify an individual among a series of registered subjects. If the system finds a match, it returns the user's identity; otherwise, identification fails.

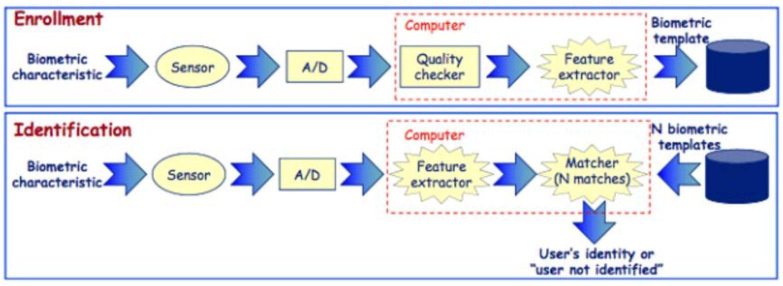


Figure 1: Outline of the identification process in a biometric system

VI. **Authentication Process:** process in which the system checks whether the biometric data provided by the user matches those previously registered. In this process, the user declares their identity (for example, an ID or a username) and proceeds with the acquisition and extraction of characteristics, similar to the identification process. A "1 to 1" comparison is made between the extracted biometric data and the template saved in the database to verify if they correspond to the declared identity, determining whether authentication has succeeded or failed.

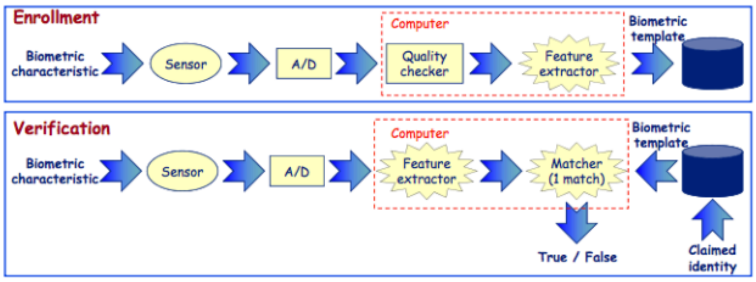


Figure 2: Outline of the authentication process in a biometric system

VII. **Biometric Template**: digital representation of distinctive characteristics extracted from a biometric sample.

VIII. **Fixed Threshold**: A default value used in decision-making systems to determine whether or not a piece of data exceeds a certain limit. In biometrics and other fields, it is employed to establish the critical point beyond which a measure, such as a biometric similarity or security level, is accepted or rejected. Unlike an adaptive threshold, a fixed threshold does not vary based on specific conditions or data, making it simple to implement but potentially less flexible in dynamic environments.

IX. **Performance Evaluation**: In biometric systems, it refers to the process of measuring the accuracy and reliability of the system in correctly recognizing users. This assessment is based on a matching score, which represents the similarity between two fingerprints, and is compared to a fixed threshold to determine whether the comparison is considered a match (matching) or not (not-matching).

X. **Interclass Similarity**: degree of similarity between biometric samples of different individuals. High interclass similarity can reduce the effectiveness of the biometric system, increasing the risk of false positives (False Acceptance Rate, FAR).

XI. **Intraclass Variability**: Variations in biometric samples of the same individual, which may be due to factors such as environmental conditions, physiological changes, or modes of acquisition. High intraclass variability can increase the False Rejection Rate (FRR) and reduce the reliability of biometric recognition.

XII. **Text dependent voice recognition:** a method of biometric voice recognition in which the text spoken by the user must be identical during both the enrollment and verification phases. In text-dependent systems, the phrase required can be the same for all users (for example, a common voice password) or customized for each individual. This approach improves the security and reliability of recognition by reducing the risk of speech imitation.

XIII. **Microcontroller:** An integrated device that combines a microprocessor with a variety of internal peripherals, such as timers, serial ports, general-purpose input/output (I/O) pins, counters, analog-to-digital converters (ADCs), program memory, and data memory, all within a single silicon chip. Due to their compact architecture and the presence of integrated components, microcontrollers are particularly suitable for embedded systems, allowing you to reduce circuit space and overall device size without the need for additional components on the board.

XIV. **Microphone:** acoustic-electric transducer that captures sound waves with a thin, flexible diaphragm and converts vibrations into an electrical signal by various methods.

XV. **Electret Microphone:** a type of condenser microphone that uses a permanently charged dielectric material, called electret, to generate the electric field necessary for its operation. This feature eliminates the need for an external power supply for polarization, making the microphone more compact, cost-effective, and suitable for a wide range of applications.

XVI. **Nyquist's theorem:** guarantees that a sampled signal can be reconstructed without loss of information, as long as the sampling frequency is at least twice the maximum frequency present in the signal, avoiding aliasing. For loss-free sampling:

$$f_s \geq 2BW$$

XVII. **Hamming Window:** A specific type of window function used to smooth out discontinuities at the edges of a signal in the frequency domain, divided into time segments.

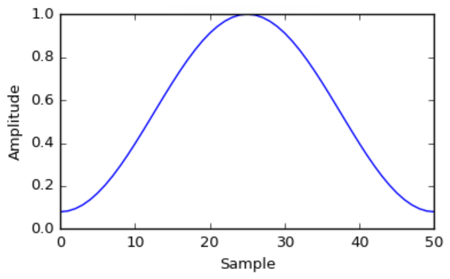


Figure 3 : Hamming Window

XVIII. **Fourier transform (FFT):** converts the signal from the time domain to the frequency domain, decomposing it into its sinusoidal components and revealing the spectral distribution of energy, an essential step for the subsequent conversion on the Mel scale.

$$x_k = \sum_{m=0}^{n-1} x_m \cdot e^{-\frac{j2\pi km}{n}} \qquad k = 0, \dots, n-1$$

XIX. **Mel scale:** a frequency scale in which frequencies are linearly spaced below 1 kHz and logarithmically above 1kHz. It is a scale of perception of the pitch of a sound that reflects the sensitivity of human hearing to frequencies: our auditory system perceives more subtle differences at low frequencies than at high frequencies.

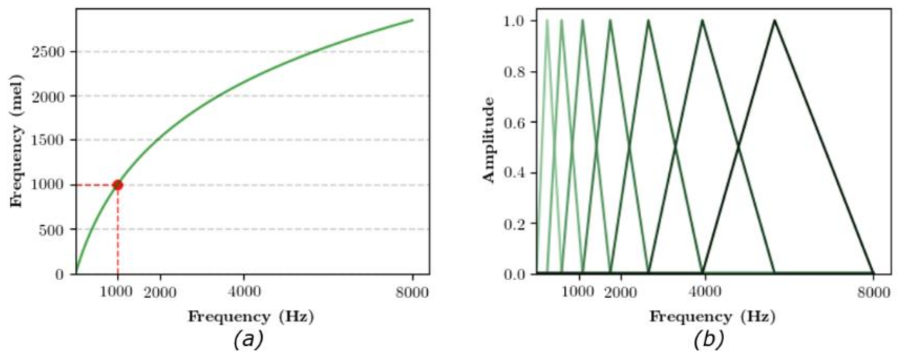


Figure 4: (a) Relationship between the linear frequency scale (Hz) and the Mel scale (mel); (b) Mel-Scale Filter Bank Composed Of Seven Filters

$$f_{Mel} = 2595 \log_{10}(1 + \frac{f_{Hz}}{700})$$

XX. **MFCC:** They are a set of numerical coefficients used to represent the power spectrum of an audio signal on a frequency scale that mimics human perception of sound. They are widely used in speech processing, speech recognition, and sound analysis due to their ability to extract relevant features from voice and other acoustic signals. The calculation of MFCCs is based on the Mel scale.
The process of extracting MFCCs follows a series of fundamental steps, which include key concepts of signal processing, including Nyquist's Theorem, Fourier transform (FFT), and Hamming's window.

XXI. **Cepstrum:** after the application of the Mel filter bank, the logarithm of the spectrum obtained is transformed using the Discrete Cosine Transform (DCT) to obtain the cepstrum, which represents the separation between the global spectral structure and the harmonic components. The cepstrum on the Mel scale is what actually makes up the MFCCs, allowing the timbre characteristics of the signal to be extracted, reducing the correlations between frequencies and obtaining a more compact and informative representation for speech analysis.

XXII. **CodeBook:** A finite set of codewords that represents a compact form of data representation. The codebook helps reduce computational complexity and data size by mapping each feature frame into a limited number of predefined prototypes.

XXIII. **CodeWord:** is a single representative vector within a feature space. Each codeword is a reference point within the codebook and is used to quantize a measured MFCC vector. The codeword is also called the centroid and is obtained as the midpoint between the cepstrums.

XXIV. **Euclidean Distance:** measure of the distance between two points in a Euclidean space. It is defined as the square root of the sum of the squares of the differences between the corresponding coordinates of the two points.

$$d(a,b) = \sqrt{\sum_{i=0}^{n-1} (b_i - a_i)^2}$$

XXV. **PCA:** dimensionality reduction technique that transforms related features into new unrelated features called principal components, sorted in descending order by variance. It uses the decomposition of the covariance matrix to identify the directions of maximum variation in the data, projecting it onto a new, small-sized space, preserving essential information.

XXVI. **Min-Max Normalization:** rescales data within a specific range, typically [0,1] or [-1,1], by using the minimum and maximum values of the dataset. The formula is:

$$\boldsymbol{X' = \frac{X - X_{min}}{X_{max} - X_{min}}}$$

where $\boldsymbol{X}$ is the original value, $\boldsymbol{X_{min}}$ is the minimum value in the dataset, and $\boldsymbol{X_{max}}$ is the maximum value. This method is useful when preserving the relative relationships of the data is important, and it is particularly effective when the dataset has a known and bounded range. However, it is sensitive to outliers, as extreme values can compress the rest of the data.

XXVII. **Z-Score Normalization:** also known as standardization, transforms a dataset so that it has a mean of zero and a standard deviation of one. It is achieved by subtracting the mean and dividing by the standard deviation of the dataset:

$$\boldsymbol{X' = \frac{X - \mu}{\sigma}}$$

where $\boldsymbol{X}$ is the original value, $\boldsymbol{\mu}$ is the mean, and $\boldsymbol{\sigma}$ is the standard deviation. This method is useful when data needs to be compared in terms of standard deviations from the mean and is effective for datasets that follow a normal distribution.

1 OBJECTIVES OF THE REPORT

The aim of this report was to document in detail the activities carried out during the laboratory sessions, dedicated to the experimentation of pre-processing techniques and recognition of electronic signals and the analysis of biometric data. The focus was in particular on the recognition of the speaker.

The laboratory experience included several phases with different objectives:

- **Voice sample acquisition**: Capture voice samples from multiple subjects using an electret microphone in a noisy environment to simulate realistic conditions, with the highest possible sampling rate.
- **Application of pre-processing techniques:** Apply speech signal pre-processing techniques in order to improve the quality of recordings and reduce the impact of distortions. Among these techniques, strategies were adopted for noise removal, normalization of signal amplitude and the application of filters to highlight the most significant components of the voice.
- **Feature extraction:** analyze the unique characteristics extracted from each subject's voice, using the Mel scale to define MFCCs (Def. XX).
- **Matching:** evaluate the accuracy of the recognition system by comparing the recorded voice samples with those present in the database.

2 HARDWARE

2.1 Components

2.1.1 STM32 Nucleo F091RC

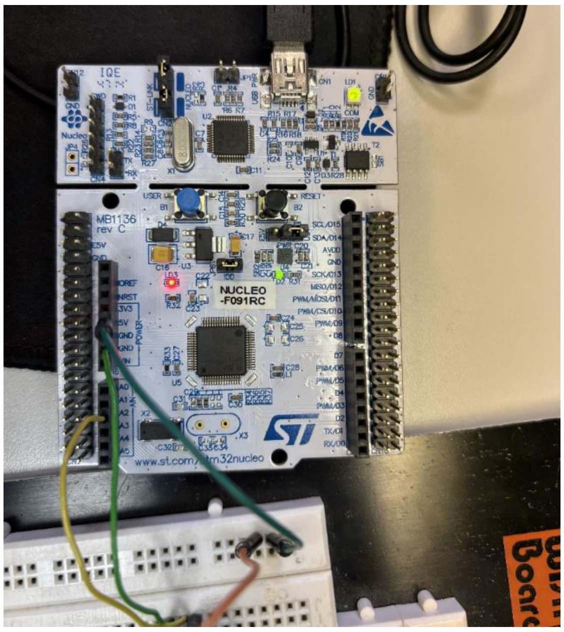


Figure 5: STM32 NUCLEO F091RC

The STM32 Nucleo-64 board provides users with a cost-effective and flexible way to experiment with new concepts and build prototypes by choosing from the various combinations of performance and power consumption offered by the STM32 microcontroller.

LEDs:

- LD1 (USB communication): Indicates the USB communication status, which is useful for viewing data activity or connections when debugging or using the Virtual COM port.
- LD2 (User LED): a customizable LED available to the user. It can be managed via software to report particular states of the application (e.g. error reporting, calculation activity, diagnostic blink, etc.).
- LD3 (Power LED): provides an indication of the power supply of the board, allowing you to quickly understand if the board is correctly powered.

BUTTONs:

- USER: a programmable button that the user can read via software as input (e.g. to implement start/stop functions, mode selection or other events).
- RESET: allows you to reset the microcontroller returning it to its initial state. It is useful both in the development and debugging phases (quick reboot) and in situations where the software needs a hard reset.

BOARD CONNECTORs:

- Arduino Uno V3 connectivity:
  - Layout compatible with standard Arduino shields, simplifying the addition of existing modules or extensions (e.g. motor shields, Wi-Fi, various sensors...).

- ST morpho extension pin headers:
    - Provides full access to all microcontroller I/O pins.
    - Useful for those who need advanced features and want to take advantage of every single peripheral of the chip (e.g. DACs, multiple ADCs, serial interfaces, etc.).
    - Allows integration with ST-specific extension boards (e.g. shields or expansion boards).

FLEXIBLE BOARD POWER SUPPLY:

- ST-LINK USB V_{BUS}: using the USB port connected to the PC, the microcontroller is powered directly by the 5 V provided by the USB cable.
- External source (3.3 V, 5 V, 7 - 12 V): If your design requires more power or a different power supply, you can connect external sources with different voltage ranges.
- Power management access point: there is a test point dedicated to monitoring and measuring consumption.

USB INTERFACEs:

- Virtual COM port: allows you to communicate with the microcontroller via a virtual serial port, useful for logging, debugging, data exchange with the application on PC (e.g. serial terminals, PuTTy).
- Mass storage: the board is recognized as a storage unit (drive). It's a quick way to load firmware onto the microcontroller: simply drag and drop the binary or .hex file into the dedicated folder.
- Debug port: using the integrated ST-LINK programmer/debugger (on board), it is possible to program and debug the microcontroller directly from the IDE (e.g. KeilStudioCloud, or others) without the need for external programmers.

HAL SOFTWARE:

- HAL library: Simplifies access to peripherals through high-level functions, reducing programming complexity. Ready-to-use starting software is available to test functionality (e.g. reading sensors, managing displays, etc.).

2.1.2 Sparkfun Electret Microphone Breakout – CMA544PF With Amplifier - MAX9814

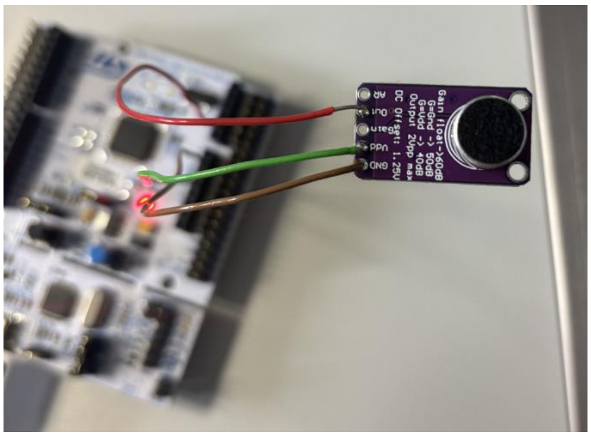


Figure 6: Sparkfun Electret Microphone

The CMA-544PF-W electret microphone is an analog electronic component designed for capturing audio signals through a permanently polarized condenser transducer. The output of the module is analog, so the analog-to-digital converter (ADC) of the microcontroller is required to obtain digital signals.

This microphone stands out for its high sensitivity and frequency response suitable for multiple audio applications, including voice capture, recording systems, and acoustic recognition devices.

It is an electret condenser microphone with a diameter of 9.7 mm and a height of 4.5 mm. It features omnidirectional directivity and a typical sensitivity of -44 dB (±2 dB) at 1 kHz, with a frequency response range of 20 Hz to 20 kHz. The output impedance is less than 2.2 kΩ, while the rated operating voltage is 3V DC, with an operating maximum of 10V DC. The maximum current consumption is 0.5 mA and the minimum signal-to-noise ratio (SNR) is 60 dB. The microphone has wire terminals for signal and ground connection.

Within the microphone there is amplifier - MAX9814 that is used to amplify the signal generated by the microphone which has very low amplitude.

Structurally, the CMA-544PF-W consists of an electret membrane and an integrated FET, which acts as a preamplifier to convert the capacitance change of the membrane into an amplified electrical signal. The electret capacitor ensures constant polarization, avoiding the need for a dedicated external power source. The omnidirectional configuration allows the microphone to capture sound from all directions evenly, making it ideal for applications where directionality is not a critical factor.

2.2 Circuits

2.2.1 Microphone

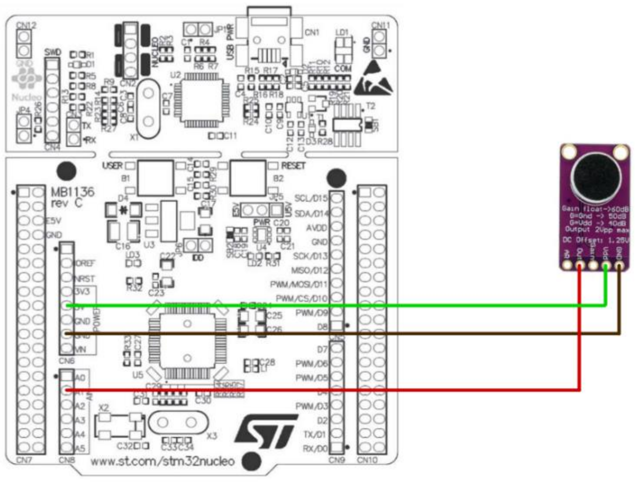


Figure 7: Circuit Diagram For Speech Acquisition

To acquire audio signals via a microcontroller, an electret microphone was used.

The connection between the microphone and the microcontroller was made as follows:

- **Power supply**: The VDD terminal of the microphone has been connected to the microcontroller's 5V power line, while the GND terminal has been connected to the common ground of the system.
- **Output signal:** The analog output of the microphone has been connected to pin A1 of the microcontroller, which acts as an analog input for digitizing the signal via the ADC (Analog-to-Digital Converter).

3 FIRMWARE

3.1 MBED OS

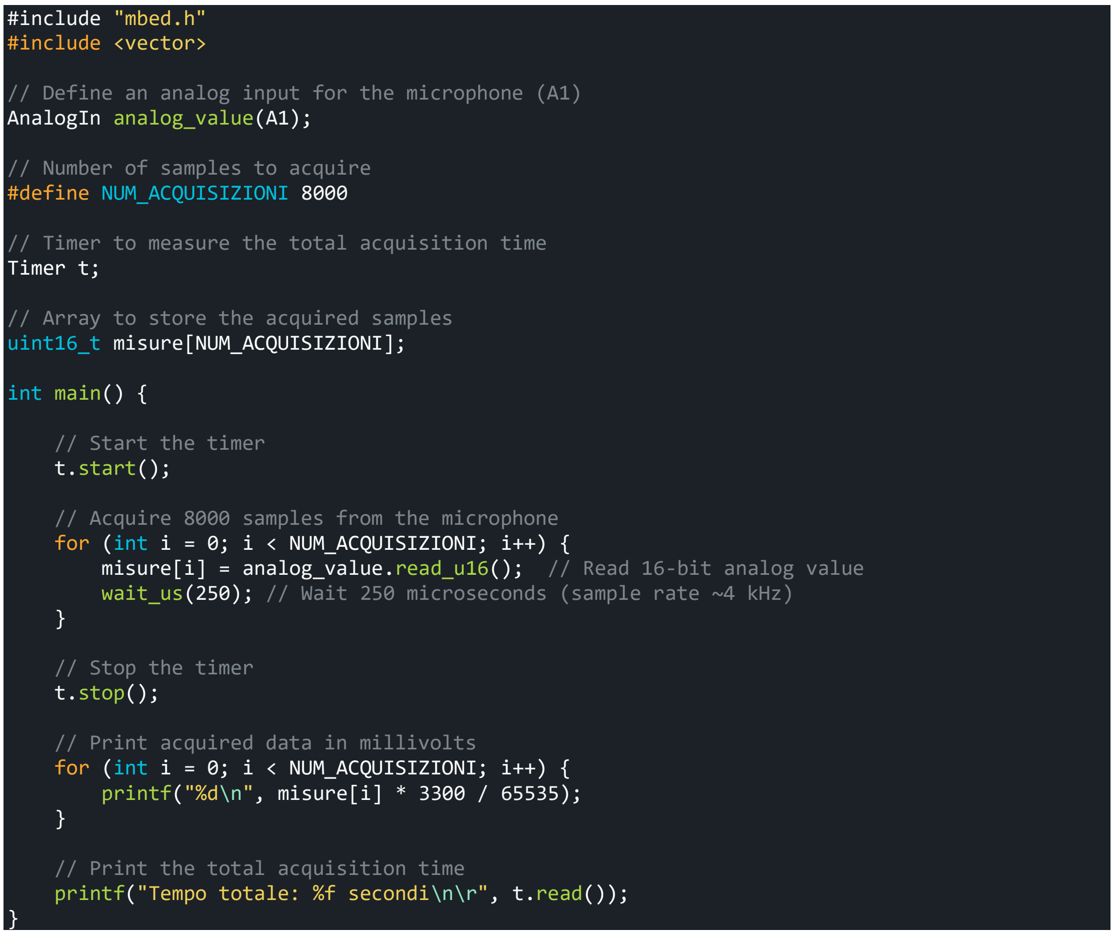
Mbed OS is an RTOS operating system developed by Arm and designed for embedded devices based on Cortex-M microcontrollers. Provides a complete, IoT-optimized development environment, with support for connectivity, security, and efficient energy management.

This operating system offers an RTOS kernel that allows for multitasking task management, thread synchronization, and efficient use of system resources. It adopts an event-driven programming model, which is especially suitable for low-power devices. It also integrates network stacks and communication protocols such as IPv6, TLS/DTLS, MQTT, and CoAP, making it easy to connect with cloud services and edge devices. The native file system manager supports several formats, including FAT and LittleFS, which are optimized for use with flash memory. Communication security is ensured by the Mbed TLS encryption library, which is designed to operate on resource-constrained devices.

Mbed OS is compatible with a wide range of development boards based on Arm Cortex-M microcontrollers and offers an ecosystem of ready-to-use drivers for different peripherals and sensors. Integration with the Mbed Cloud framework facilitates remote deployment and management of IoT devices.

Keil Studio is the IDE provided by Arm for programming embedded applications on Mbed OS. This platform includes advanced tools for writing, compiling, and debugging code, with support for CMSIS (Cortex Microcontroller Software Interface Standard) and the Arm libraries for Cortex-M microcontrollers. The system supports programming in C/C++ and offers advanced real-time debugging, simulation, and hardware emulation capabilities. With the integration of Arm Compiler 6, it enables the generation of optimized code to reduce power consumption and improve performance. In addition, the implementation of Keil RTX RTOS ensures efficient thread and resource management, improving the reliability of embedded applications. The debugging and tracing system allows a detailed analysis of the software's behavior, allowing the optimization of critical applications.

3.1.1 Voice Acquisition



```
#include "mbed.h"
#include <vector>

// Define an analog input for the microphone (A1)
AnalogIn analog_value(A1);

// Number of samples to acquire
#define NUM_ACQUISIZIONI 8000

// Timer to measure the total acquisition time
Timer t;

// Array to store the acquired samples
uint16_t misure[NUM_ACQUISIZIONI];

int main() {

    // Start the timer
    t.start();

    // Acquire 8000 samples from the microphone
    for (int i = 0; i < NUM_ACQUISIZIONI; i++) {
        misure[i] = analog_value.read_u16();  // Read 16-bit analog value
        wait_us(250); // Wait 250 microseconds (sample rate ~4 kHz)
    }

    // Stop the timer
    t.stop();

    // Print acquired data in millivolts
    for (int i = 0; i < NUM_ACQUISIZIONI; i++) {
        printf("%d\n", misure[i] * 3300 / 65535);
    }

    // Print the total acquisition time
    printf("Tempo totale: %f secondi\n\r", t.read());
}
```

Figure 8: Firmware For 16bit And 4000Hz Audio Sampling With Microphone

The code uses the Mbed library to capture and store an analog signal from a microphone connected to pin A1. The analog_value object is declared as *AnalogIn(A1),* which allows the input analog values to be read. The *constant NUM_ACQUISIZIONI* is defined as 8000 and represents the total number of samples that will be acquired.
Next, an array of measurements of type *uint16_t* and *size NUM_ACQUISIZIONI is declared*, which will be used to store the analog readings.
This choice was necessary because the *printf* operation has a duration of about 10 ms, a value comparable with the sampling time. As a result, a delay is introduced that compromises the correct acquisition of data.
Within the *main* function, the *for* loop iterates 8000 times, reading a value from the microphone with *analog_value.read_u16()* on each iteration, which returns a 16-bit value between 0 and 65535. This value is stored in the *measurement* array, and after each reading the program waits 250 microseconds via *wait_us(250)*, thus establishing a sampling rate of about 4 kHz.
A second for loop is used to iterate through the *measurement* array and print the acquired values converted to millivolts, using the formula *measurements[i]*3300/65536*, assuming a reference of 3.3V for the analog-to-digital converter.

The first acquisitions were made starting with a sampling frequency of only 50Hz, which according to the Nyquist sampling theorem are insufficient for an error-free reconstruction of the signal and therefore a correct classification of users. We were able to increase the sample rate to 1kHz, 2kHz, and finally 4kHz with the solution presented above, thus using 16-bit

samples. The sample rate could not be increased beyond due to the small memory size. We could have reduced the sample size to 8 bits to support a higher frequency, at 8kHz. The choice fell on the 4kHz option because the higher fidelity in quantization and therefore greater precision in the nuances of volume (wider dynamic range) was preferred.
By choosing, instead, an 8kHz sampling rate we would have obtained better presentation of the frequencies of the voice given the frequency range of the human voice.

3.2 Matlab

MATLAB is a programming environment and software developed by MathWorks, widely used for numerical computing, simulation, and data visualization.

This software is optimized for matrix and vector operations and supports advanced mathematical functions such as linear algebra, interpolation, Fourier transforms, differential equations, and optimization. The interactive programming environment offers an intuitive user interface with a script editor and command window, allowing for both structured programming and interactive experimentation.

One of the distinguishing features of MATLAB is the presence of numerous specialized toolboxes that expand its capabilities in different application areas. These include Signal Processing Toolbox, for signal analysis and filtering, Fourier transforms (FFT), and FIR/IIR filter design, Statistics and Machine Learning Toolbox, statistical analysis, predictive modeling, clustering, and machine learning, Audio Toolbox, for filtering, audio effects, voice analysis, audio feature extraction (MFCC, pitch, formants).

In addition, MATLAB allows you to create advanced 2D and 3D graphs with extensive customization possibilities, making data representation and analysis particularly effective.

3.2.1 Speaker Recognition

```
clc;
clear all;
close all;

rng(123);  % Fix the seed for randomization

% List of Excel files (for each user and noise)
files = {
    'utente_1_Damato.xlsx',
    'utente_2_Damato.xlsx',
    'utente_3_Damato.xlsx',
    'rumore.xlsx'
};
% Parameters
fs = 4000; % Sampling frequency (Hz)
dt = 1 / fs; % Sampling interval (s)

% Pre-allocation for data
segnali = {};  % Cell array for user signals
rumore = [];   % Variable for noise signal
% Reading data from each file
for i = 1:length(files)
    % Read the Excel file
    data = readmatrix(files{i});
        % Save data in a cell array (signals from users)
    if i < 4
        segnali{i} = data;
    else
        rumore = mean(data, 2);  % Calculate the average noise (mean across columns)
    end
end

% Number of acquisitions per file (9 for users, 3 for noise)
num_acquisizioni = size(segnali{1}, 2);

% Threshold parameters and processing settings
soglia_relativa = 0.03;
gamma = double(intmax('int16'));
soglia = soglia_relativa * gamma;
soglia_relativa2 = 0.71;

% Pre-allocation for final signals
segnali_finali = cell(1, 9);
indice_salvataggio = 1;

% Signal Processing
segnali_senza_rumore = cell(length(segnali)-1, num_acquisizioni);
segnali_prima_soglia = cell(length(segnali)-1, num_acquisizioni);
t_utile_graph = cell(length(segnali)-1, num_acquisizioni);

% Signal processing loop
for i = 1:length(segnali)
    % Create a new figure for each file
    figure('Name', ['Segnali - ' files{i}(1:end-5)], 'NumberTitle', 'off');

    % Time axis
    N = size(segnali{i}, 1);
    t = (0:N-1) * dt;

    % Process signals
    for j = 1:num_acquisizioni
        subplot(num_acquisizioni, 1, j);  % Subplot for each acquisition

        % Calculate clean signal by subtracting noise
        segnale_pulito = segnali{i}(:, j) - rumore;
        non_silenzio = abs(segnale_pulito) > soglia;
        inizio = find(non_silenzio, 1, 'first');
        fine = find(non_silenzio, 1, 'last');
```

```
            if ~isempty(inizio) && ~isempty(fine)
             % Extract useful signal
             segnale_utile = segnale_pulito(inizio:fine);
             t_utile = t(inizio:fine);
             gamma2 = max(abs(segnale_utile));
             soglia2 = soglia_relativa2 * gamma2;

             % Plot original and useful signals
             plot(t, segnale_pulito, 'b', 'DisplayName', 'Original Signal');
             hold on;
             plot(t_utile, segnale_utile, 'r', 'DisplayName', 'Useful Signal');

             % Apply second threshold
             non_silenzio2 = abs(segnale_utile) > soglia2;
             inizio2 = find(non_silenzio2, 1, 'first');
             fine2 = find(non_silenzio2, 1, 'last');

             if ~isempty(inizio2) && ~isempty(fine2)
                 segnale_finale = segnale_utile(inizio2:fine2);
                 t_utile2 = t_utile(inizio2:fine2);

                 plot(t_utile2, segnale_finale, 'g', 'DisplayName', 'Final Signal (second
                 threshold)');

                 % Save final signal
                 if indice_salvataggio <= 9
                     segnali_finali{indice_salvataggio} = segnale_finale;
                     indice_salvataggio = indice_salvataggio + 1;
                 end
             end
         else
             % No useful signal: plot an empty line
             plot(t, zeros(size(t)), 'b');  % Empty line
             title(['No useful signal for acquisition ' num2str(j) ' - ' files{i}(1:end-5)]);
         end

         % Customize the plot
         grid on;
         title(['Signal - Acquisition ' num2str(j) ' - ' files{i}(1:end-5)]);
         xlabel('Time [s]');
         ylabel('Amplitude [mV]');
         % Add separate legend objects for each user
         s1 = plot(NaN, NaN, '-b', 'LineWidth', 2);
         s2 = plot(NaN, NaN, '-r', 'LineWidth', 2);
         s3 = plot(NaN, NaN, '-g', 'LineWidth', 2);
         legend([s1, s2, s3], 'Denoised Signal', 'Speech Segment (1° Threshold)', 'Speech Segment
(2° Threshold)');
         hold off;

    end
end

% Z-Score normalization for all signals
num_segnali = length(segnali_finali);
for i = 1:num_segnali
    segnali_finali{i} = (segnali_finali{i} - mean(segnali_finali{i})) /
std(segnali_finali{i});
end

% Select signals to add to the two new vectors
segnali_finali_train = {segnali_finali{1}, segnali_finali{2}, segnali_finali{4},
segnali_finali{5}, segnali_finali{7}, segnali_finali{8}};
segnali_finali_test = {segnali_finali{3}, segnali_finali{6}, segnali_finali{9}};
% Display the length of both groups
disp('First group (segnali_finali{1}, segnali_finali{2}, segnali_finali{4},
segnali_finali{5}, segnali_finali{7}, segnali_finali{8}):');
disp(length(segnali_finali_train));
disp('Second group (segnali_finali{3}, segnali_finali{6}, segnali_finali{9}):');
disp(length(segnali_finali_test));
```

```
% Assume signals are stored in segnali_finali{1} ... segnali_finali{9}
% Each cell contains a different signal of variable length
% Pre-processing parameters
frame_length = 256;   % Frame length (number of samples)
frame_overlap = 128;  % Frame overlap
fs = 4000;            % Sampling frequency

% Initialize the vector to store MFCCs
mfcc_all_users = cell(1, 6);

% Calculate MFCCs for training signals
for i = 1:6
    % Load the signal
    signal = segnali_finali_train{i};

    % Calculate MFCC for each signal
    % Use MATLAB's mfcc function to calculate the MFCCs
    mfcc_all_users{i} = mfcc(signal, fs, 'Window', hamming(frame_length), 'OverlapLength',
frame_overlap, 'NumCoeffs', 13);
end

% Display the size of each MFCC matrix
for i = 1:6
    % Get the MFCC matrix for each signal
    mfcc_matrix = mfcc_all_users{i};

    % Store the dimensions (number of frames and coefficients)
    dimensions(i, :) = size(mfcc_matrix);
end

% Print dimensions
disp(dimensions);

% Initialize a vector to store centroids
centroidi = zeros(6, 14);  % 6 signals, 14 MFCC coefficients per centroid

% Calculate centroid for each signal
for i = 1:6
    % Get MFCC for the i-th signal
    mfcc_signal = mfcc_all_users{i};

    % Calculate the centroid as the mean across frames (along rows)
    centroide_signal = mean(mfcc_signal, 1);

    % Store the centroid
    centroidi(i, :) = centroide_signal;
end

% Define colors for each user as cell arrays of strings
colori = {'r', 'g', 'b', 'w'}; % 'r' for Antonio, 'g' for Vittoria, 'b' for Raffaele

% Perform PCA to reduce dimension from 14 to 2
[coeff, score, ~] = pca(centroidi);

% Create an array of colors for each centroid based on the user
utenti_colori = [repmat(colori{1}, 2, 1);  % 2 signals for Antonio
                 repmat(colori{2}, 2, 1);  % 2 signals for Vittoria
                 repmat(colori{3}, 2, 1)]; % 2 signals for Raffaele

% Display centroids in the new 2D space
figure;
hold on;

% Plot each centroid with the corresponding user color
for i = 1:6
    scatter(score(i, 1), score(i, 2), 100, 'MarkerFaceColor', utenti_colori(i,:),
'MarkerEdgeColor', 'k', 'LineWidth', 1.5);
end

% Add objects for the legend separated by user
h1 = scatter(NaN, NaN, 100, 'MarkerFaceColor', colori{1}, 'MarkerEdgeColor', 'k'); % Antonio
h2 = scatter(NaN, NaN, 100, 'MarkerFaceColor', colori{2}, 'MarkerEdgeColor', 'k'); % Vittoria
h3 = scatter(NaN, NaN, 100, 'MarkerFaceColor', colori{3}, 'MarkerEdgeColor', 'k'); % Raffaele
```

```matlab
% Add legend and details
title('Speaker Codebook with Centroids for each User (Enrollment)');
xlabel('Principal Components 1');
ylabel('Principal Components 2');
grid on;
legend([h1, h2, h3], 'Antonio', 'Vittoria', 'Raffaele');
hold off;

%%
% Threshold for deciding membership (define based on your data)
soglia = 0.7; % Change this value depending on the scale of distances
% Preallocate a cell to store the centroids of the new signals
centroidi_nuovi_segnali = zeros(3, 14); % 3 new signals, 14 MFCC coefficients

% Calculate the centroids for each new signal
for j = 1:3
    % Compute the MFCCs of the new signal
    mfcc_nuovo_segnale = mfcc(segnali_finali_test{j}, fs, 'Window', hamming(frame_length),
'OverlapLength', frame_overlap, 'NumCoeffs', 13);
    % Calculate the centroid of the new signal
    centroidi_nuovi_segnali(j, :) = mean(mfcc_nuovo_segnale, 1);
end

% Compute the distance between the centroids of the new signals and the existing centroids
distanze = zeros(3, 6); % 3 new signals and 6 centroids (3 for each user)

% Calculate the Euclidean distances for each new signal
for j = 1:3
    for i = 1:6
        distanze(j, i) = norm(centroidi_nuovi_segnali(j, :) - centroidi(i, :));
    end
end

% Matrix to store the average distances for each user
medie_distanze = zeros(3, 3); % 3 new signals, 3 users

% Calculate the average distance for each user
for j = 1:3
    % Distance from the centroids of 'Antonio' (1 and 2)
    medie_distanze(j, 1) = mean(distanze(j, 1:2));
    % Distance from the centroids of 'Vittoria' (3 and 4)
    medie_distanze(j, 2) = mean(distanze(j, 3:4));
    % Distance from the centroids of 'Raffaele' (5 and 6)
    medie_distanze(j, 3) = mean(distanze(j, 5:6));
end

% Display the average distances for each user
disp('Average distances for each user:');
disp(medie_distanze);

% Classify the new signals based on the minimum distance from the averages
utenti_classificati = cell(1, 3); % Store the user name or "None" for each new signal
utenti_reali = {'Antonio', 'Vittoria', 'Raffaele'}; % Actual users of the new signals
segnali_correttamente_classificati = 0; % Counter for correctly classified signals
segnali_classificati = 0; % Counter for signals assigned to a user

for j = 1:3
    % Find the minimum distance and the index of the candidate user
    [distanza_minima, indice_utente] = min(medie_distanze(j, :));
    disp(distanza_minima)

    % Check if the minimum distance is below the threshold
    if distanza_minima < soglia
        % Assign the new signal to the corresponding user
        if indice_utente == 1
            utenti_classificati{j} = 'Antonio';
        elseif indice_utente == 2
            utenti_classificati{j} = 'Vittoria';
        else
            utenti_classificati{j} = 'Raffaele';
        end
        % Check if the classification is correct
        if strcmp(utenti_classificati{j}, utenti_reali{j})
            segnali_correttamente_classificati = segnali_correttamente_classificati + 1;
        end
    else
        % Classify the new signal as "None"
        utenti_classificati{j} = 'Nessuno';
    end
end

% Calculate the accuracy only for the signals that were classified
accuracy = segnali_correttamente_classificati / length(segnali_finali_test);

% Display the accuracy
disp(['The classification accuracy is: ', num2str(accuracy * 100), '%']);

% Display the classification result
for j = 1:3
    disp(['New signal ', num2str(j), ' is classified as belonging to: ',
utenti_classificati{j}]);
end

%%
% Combine all centroids (initial and new) into a single matrix
tutti_centroidi = [centroidi; centroidi_nuovi_segnali];

% Perform PCA to reduce dimensionality to 2D
[coeff, score, ~] = pca(tutti_centroidi);

% Project the centroids into the 2D space
proiezioni_centroidi_iniziali = score(1:6, :); % The first 6 initial centroids
proiezioni_centroidi_nuovi = score(7:end, :);  % The last 3 new centroids

% Colors for the new centroids based on the identified users
colori_nuovi = cell(1, 3); % Preallocate the colors for the new centroids

for j = 1:3
    if strcmp(utenti_classificati{j}, 'Antonio')
        colori_nuovi{j} = 'r'; % Red
    elseif strcmp(utenti_classificati{j}, 'Vittoria')
        colori_nuovi{j} = 'g'; % Green
    elseif strcmp(utenti_classificati{j}, 'Raffaele')
        colori_nuovi{j} = 'b'; % Blue
    elseif strcmp(utenti_classificati{j}, 'Nessuno')
        colori_nuovi{j} = 'w'; % White
    end
end

% Create the plot
figure;
hold on;

% Plot the initial centroids with black outline
for i = 1:3
    % Indices of the centroids for user i
    indici_iniziali = (2 * i - 1):(2 * i);
    scatter(proiezioni_centroidi_iniziali(indici_iniziali, 1), ...
            proiezioni_centroidi_iniziali(indici_iniziali, 2), ...
            100, colori{i}, 'o', 'filled', ...
            'MarkerEdgeColor', 'k', 'LineWidth', 1.5);
end

% Plot the new centroids with black outline
for j = 1:3
    scatter(proiezioni_centroidi_nuovi(j, 1), ...
            proiezioni_centroidi_nuovi(j, 2), ...
            150, colori_nuovi{j}, '^', 'filled', ...
            'MarkerEdgeColor', 'k', 'LineWidth', 1.5);
end
```

```
% Add objects for the legend separated for each user
p1 = scatter(NaN, NaN, 100, 'MarkerFaceColor', colori{1}, 'MarkerEdgeColor', 'k'); % Antonio
p2 = scatter(NaN, NaN, 100, 'MarkerFaceColor', colori{2}, 'MarkerEdgeColor', 'k'); % Vittoria
p3 = scatter(NaN, NaN, 100, 'MarkerFaceColor', colori{3}, 'MarkerEdgeColor', 'k'); % Raffaele
p4 = scatter(NaN, NaN, 100, 'MarkerFaceColor', colori{4}, 'MarkerEdgeColor', 'k'); % None

% Add example points for shapes with black outline
p5 = scatter(NaN, NaN, 100, 'o', 'MarkerEdgeColor', 'k', 'LineWidth', 1.5); % Circle with
black outline
p6 = scatter(NaN, NaN, 100, '^', 'MarkerEdgeColor', 'k', 'LineWidth', 1.5); % Triangle with
black outline

% Add the legend with required specifications
legend([p1, p2, p3, p4, p5, p6], 'Antonio', 'Vittoria', 'Raffaele', 'None', 'Initial
centroids', 'New centroids');

% Configure the plot
xlabel('First principal component');
ylabel('Second principal component');
title('PCA Representation of the Centroids');
grid on;
hold off;
```

Figure 9: Code for Audio Signal Processing and User Recognition Using MFCC

The code starts with setting the seed for the random number generator using *rng(123)*, so that the results are reproducible if the code is executed multiple times. An array of cells called files is then defined, which contains the names of the Excel files from which the data will be read: these files contain both user signals and noise data. Next, some parameters are defined such as the sampling frequency *fs*, which is set at 4000 Hz, and the sampling interval *dt*, which is calculated as the reciprocal of the sampling rate.

The code allocates variables to store user signals and noise. *segnali* is a cell that will house the signals of each user, while *rumore* is an array intended to hold the noise data. Subsequently, through a *for loop*, the program iterates through the files contained in the array *files* and, for each file, reads the data using the *readmatrix* function. If the file is about a user signal (the first three files in the array), the data is stored in the *signal array*. If, on the other hand, the file is about noise (the last file), the code calculates the average of the data on all the columns and stores it in *rumore*.

The program then enters a signal pre-processing phase, in which, for each signal file, a figure is generated to display the results. Within a loop, the code determines the number of samples of the signal and constructs a time axis based on the sampling rate.
For each signal acquisition, the following steps are performed:

- **Denoising the original signals.**
  To clean up the voice signal from noise, we chose to make an acquisition dedicated only to it, and then subtract it from the voice signals of each user. One of the key benefits of this technique is the ability to selectively remove noise without excessively altering the vocal content. This method can also work well when noise overlaps the vocal spectrum, where accurate separation based on frequency alone is difficult. The disadvantage is that the noise requires ad hoc acquisition, and moreover, this method only works well if the noise is stationary and well estimated. If the noise changes over time, the subtraction may be inaccurate and leave residue.
  An alternative approach would have been to use frequency filters, which are easier to implement but risk eliminating useful signal in the same frequency band as the noise, especially in the case of noise due to surrounding voices, compromising the quality of the resulting audio.

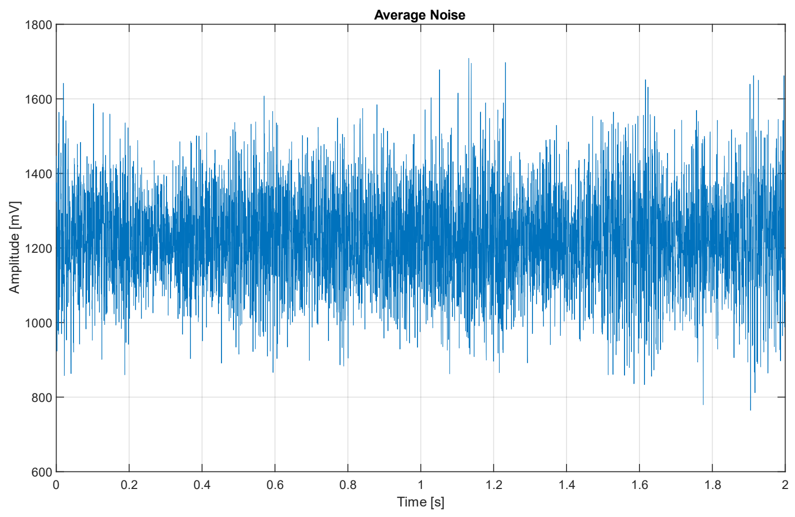


Figure 10: Noise Signal Obtained By Averaging The Three Pure Noise Acquisition

- **Truncation of speech segment using thresholds.**
  At the signal obtained from the previous phase, two thresholds are used to truncate the speech segments by eliminating silence. The first threshold, referred to as *soglia_relativa*, is set at 0.03, after several attempts, and is applied to the denoised signal to identify significant segments of the speech signal compared to the background noise. The choice of this value is dictated by the need to eliminate the parts of the signal that contain only noise, avoiding losing relevant information. To establish the absolute threshold, multiply the relative value by the maximum value that can be represented with a 16-bit integer, in order to maintain consistency with the numerical representation of the acquired data.

  Subsequently, since, for some signals, the use of the first threshold did not guarantee a high precision in the selection of the useful signal, a second threshold is applied, called *soglia_relativa2*, with a value of 0.71, which is based on the maximum amplitude of the signal obtained after the first filtering. This value was chosen, also in this case after several attempts, to remove additional low-amplitude signal sections, which could not contain significant vocal information and derive from residual noise phenomena. This double threshold was used to improve the accuracy in the selection of the useful signal, ensuring that only speech is considered for the subsequent stages of processing.

  The second threshold could have been applied directly without going through the first, but we chose to keep both to show the experimental process followed. The use of the first threshold makes the way in which the signal is progressively filtered more evident, reflecting the approach taken to optimize processing.

  The denoised signals, useful signals and final signals are then plotted on a graph, using different colors for each type of signal (blue for the original signal, red for the useful signal, green for the final signal) (*Figure 13*)(*Figure 14*)(*Figure 15*).
  Finally, the final signals, are saved in the *segnali_finali* array. A series of graphs shows the evolution of the signal, from the original version, through noise filtering, to the final signal that contains only useful information .

- **Normalization.**
  Normalization is used to make data more consistent and comparable, improving the performance of the recognition system. When data has different scales, some variables can dominate others, so normalization helps prevent this problem by ensuring that all features have a balanced impact.

Arrays saved in *segnali_finali* were normalized by trying two different normalization techniques: Z-Score (Def. XXVII) e Min–Max (Def. XXVI).
Only the first normalization technique is present in the code, having provided a higher system accuracy.

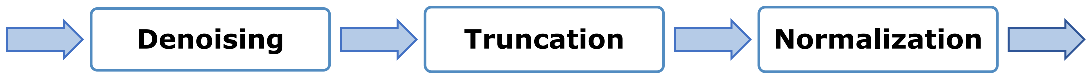


Figure 11: Steps Of The Pre-Processing Phase

Subsequently, the signals are divided into two distinct groups, a group of training signals and another of tests, for each of which the acoustic characteristics are calculated using the method of MFCCs (Mel-Frequency Cepstral Coefficients) (Def. XX). MFCCs are extracted using a 256-sample-long Hamming window with an overlap of 128 samples that allow to obtain a representation of the signals in a smaller space, useful for classification operations. MFCCs are calculated for each training signal, and the results are stored in an array of `mfcc_all_users` cells, with an index for each user.
Next, the MFCC centroid for each user is calculated, which is the average of the MFCC coefficients over all frames for each signal. These centroids are a synthetic representation of each user's acoustic characteristics and are used as the basis for a dimensional reduction operation using Principal Component Analysis (PCA) (Def. XXV), which projects data from a 14-dimensional space (the MFCC coefficients) to a two-dimensional space. This reduces complexity and allows for graphical data visualization, where each user's centroids are colored separately for easy distinction between users. (*Figure 16*)

In the code, Euclidean distances (Def. XXIV) between the centroids of the training users and the centroids of the new test signals are also calculated. The calculation of distances is used to determine to which user each new signal belongs, based on the similarity of its acoustic characteristics with those of already known users. For each test signal, the average distances between the centroid of the test signal and the centroids of the users are calculated, and the corresponding user is then assigned to the nearest centroid, as long as the distance is less than a set threshold. If the distance is greater than the threshold, the signal is classified as "No Match".

Then, the next phase is the calculation of the accuracy of the classification system, i.e. the percentage of correctly classified signals compared to the total number of signals tested. At the end of the classification process, the results are displayed in a graph that shows, via PCA, the centroids of the original users and those of the test signals, with distinctive colors for each user. Classified test signals are represented with distinct symbols (triangles), while initial centroids are shown with circles (*Figure 17*).

4 RESULTS

4.1 Speaker Recognition

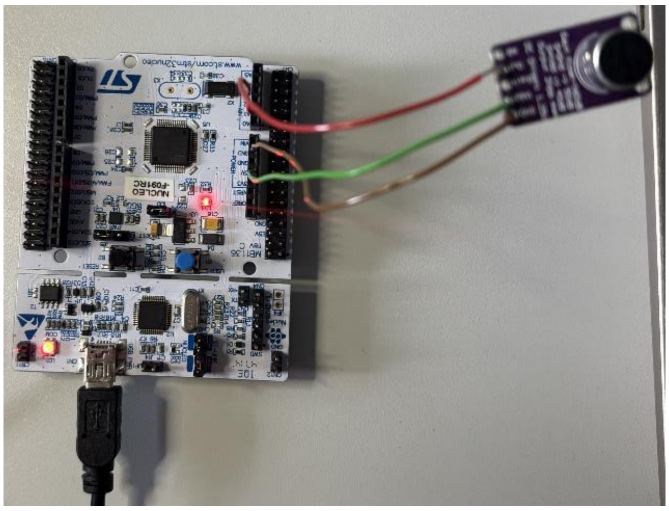


Figure 12: Microcontroller and Electret Microphone

During the laboratory experience, we worked on speech recognition to analyze the acquired voice signals and identify an individual based on the characteristics of his or her voice (Figure 9).

For this study, the acquisitions of the pronunciation of each of the participants' surnames were made by each of the participants.

In the figures below, containing all the acquisitions without the noise and truncated, three fundamental components are observed:

- The blue signal represents the denoised signal.
- The red signal is the part of the signal that passes an initial truncation phase by subtraction of silent period, based on an first threshold.
- The green signal represents the most significant part of speech obtained after a second truncation phase.

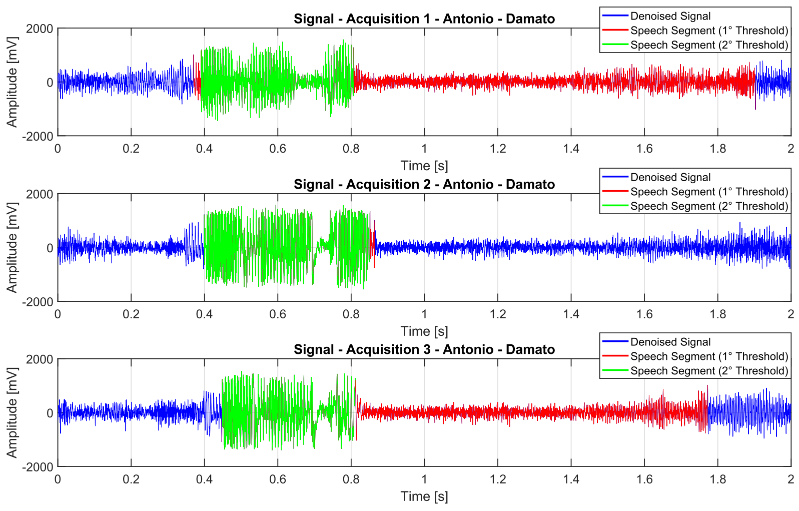
For each of the acquired vocal signals, an initial threshold was applied to remove silence, ensuring that only the relevant portions of the signal were retained. However, in certain cases, this first threshold was not sufficient to completely eliminate residual silent segments. In these instances, a second threshold was applied selectively, allowing for the removal of any remaining unwanted silence while preserving the integrity of the vocal content. This approach ensured a more refined signal processing, optimizing the clarity and usability of the acquired data.

For example, in the second acquisition of the *Figure 14 (a)*, the first threshold effectively removes the initial and final silence, leaving only the spoken content intact. Since there are no significant residual silent segments, no further processing is required, and the signal is ready for analysis.

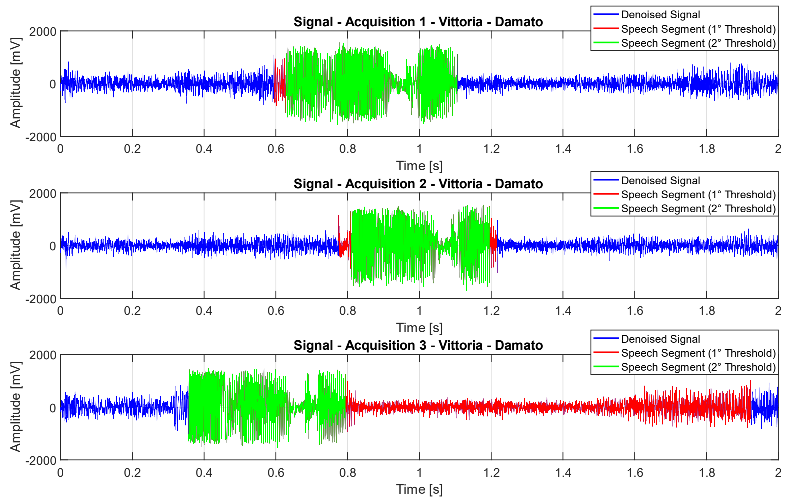
In other cases, for instance in the first acquisition of the *Figure 13 (a),* applying a second threshold becomes necessary to refine the result, ensuring that only meaningful vocal elements are preserved without disrupting the natural flow of speech.

In few cases, the second threshold might inadvertently remove a small portion of the initial or final part of the speech. This can happen when low-amplitude voiced segments are mistakenly classified as silence, leading to a slight truncation of the speech segment. This can be observed in the first acquisition of the *Figure 14 (a)*. While it would be possible to refine the threshold to better preserve these parts, doing so could risk reducing its effectiveness in cases where it initially worked well, potentially leaving residual silence in recordings that previously had no issues. Anyway, these little removed segments have no impact on the overall performances of the system.

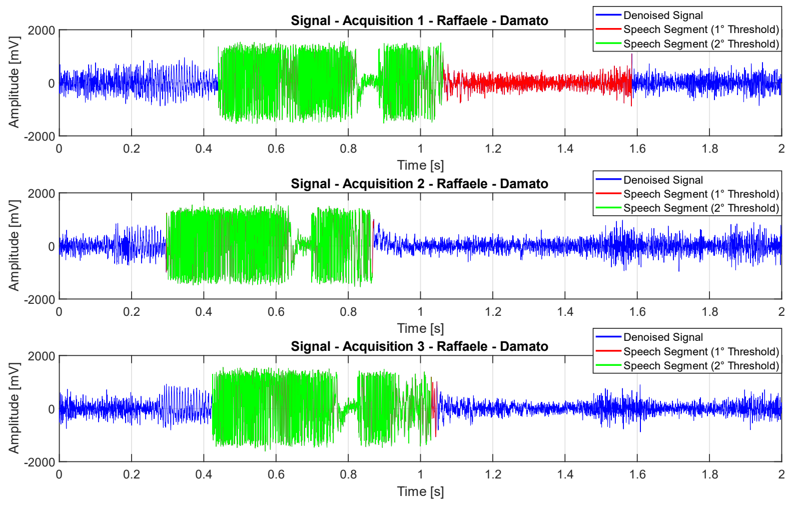
After this phase, several data normalization techniques were applied, the impact of which on the performance of the system will be explored in detail in the section on the evaluation of the results.



(a)

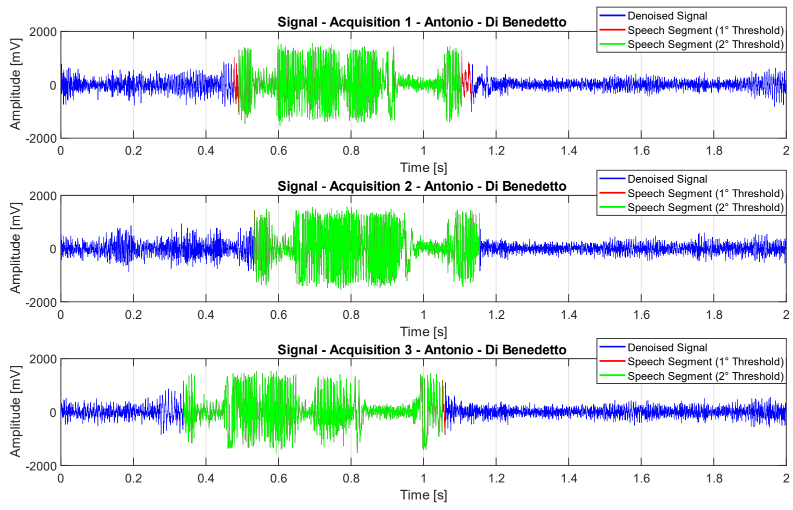


(b)

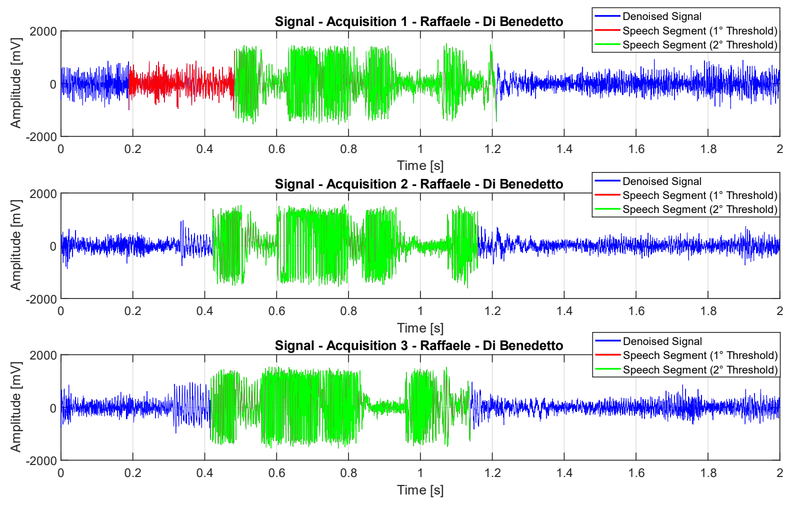


(c)

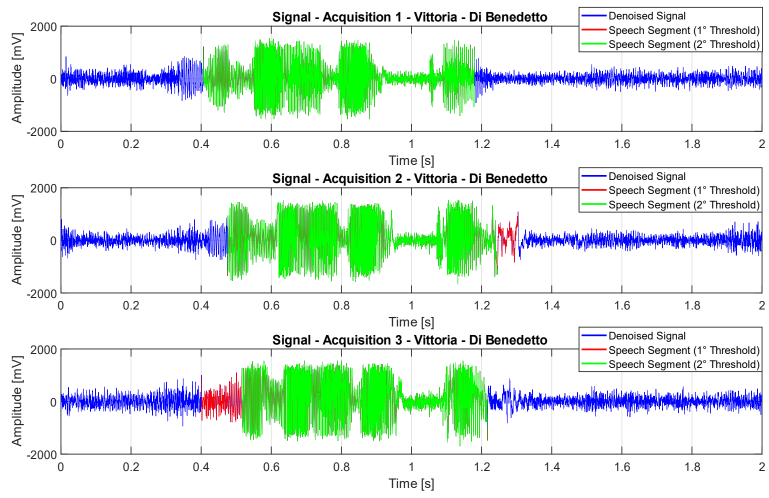
Figure 13: Denoised Signals Truncation Through Two Thresholds For Surname "Damato"



(a)

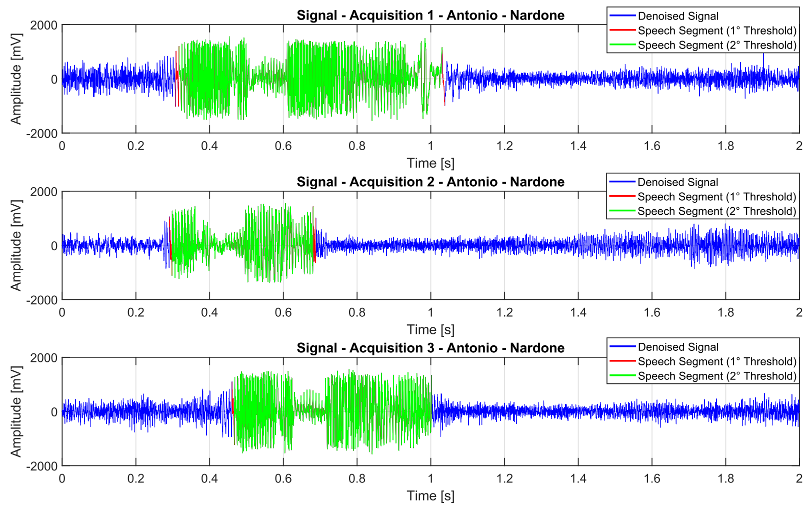


(b)

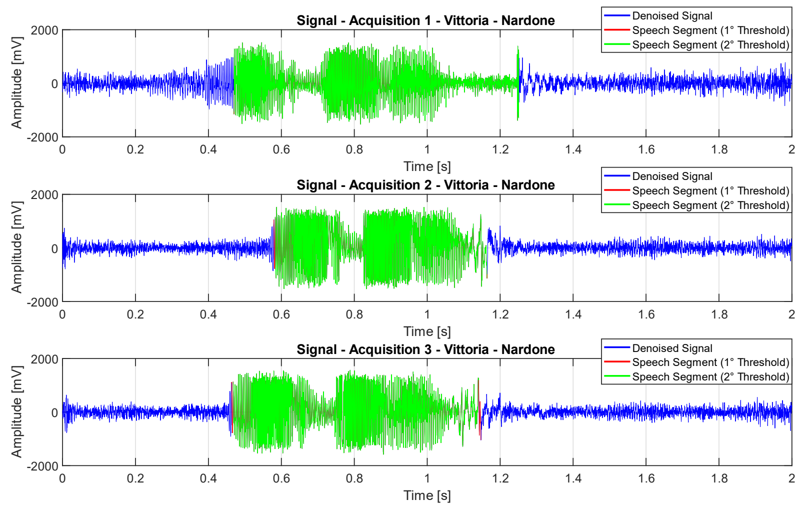


(c)

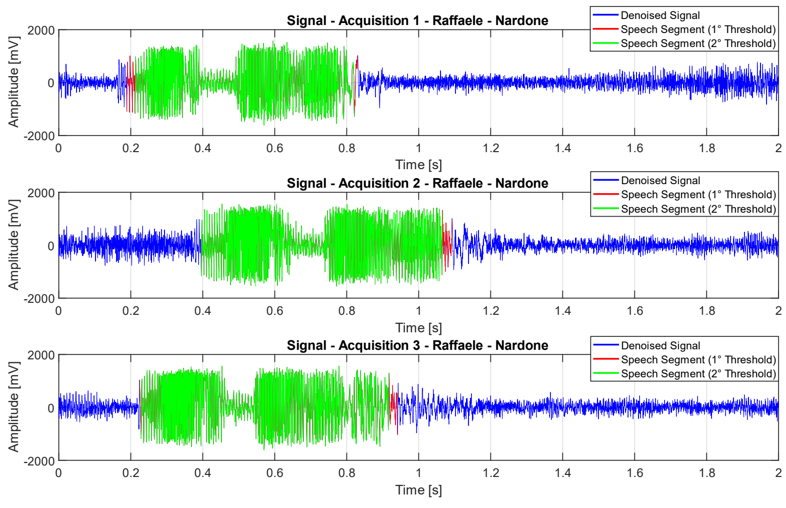
Figure 14: Denoised Signals Truncation Through Two Thresholds For Surname "Di Benedetto"



(a)



(b)



(c)

Figure 15: Denoised Signals Truncation Through Two Thresholds For Surname "Nardone"

After finishing the pre-processing phase, we moved on to the calculation of the MFCCs, in particular 13 for each speaker, through the use of the *mfcc function* of Audio Toolbox.

MFCCs will represent the features of our Speaker Recognition system.

As input parameters to the function, we have provided:

- Speech segment;
- 4kHz sampling rate;
- Hamming window specifying the length of the frame blocking (Def. XVII);
- Overlapping length;
- Number of MFCCs you want to extract.

The output of the function is a matrix containing 13 columns, equal to the number of MFCCs extracted, and a number of rows equal to the number of frames into which the signal was divided.

In the analysis of the acquired vocal signals, a **Hamming window** of 256 samples was chosen, with an overlap of 128 samples and the extraction of 13 MFCC coefficients. These choices were made to balance time and frequency resolution while ensuring robust feature extraction.

With a **sampling frequency** of 4 kHz and a total of 8000 samples, each frame of 256 samples corresponds to a duration of 64 ms (since 256 / 4000 = 0.064 s). This window length provides a good trade-off between capturing short-term speech characteristics and maintaining sufficient frequency resolution. The Hamming window was specifically selected because it reduces spectral leakage compared to a rectangular window, leading to smoother and more reliable frequency domain representations.

Choosing higher or lower values for these parameters would therefore have been suboptimal. A larger window size (e.g., 512 samples) would have improved frequency resolution but at the cost of poorer time resolution, making it harder to capture rapid changes in speech. Conversely, a smaller window (e.g., 128 samples) would have provided better time resolution but led to a lower frequency resolution, potentially making feature extraction less robust.

The **overlapping length** of 128 samples (which corresponds to 50% of the window length) ensures a good temporal resolution while maintaining continuity between frames. A lower overlap (e.g., 25%) would result in a loss of important transition details in speech, leading to poorer time resolution. On the other hand, a higher overlap (e.g., 75%) would increase computational cost without significantly improving feature quality, as adjacent frames would become highly redundant.

The choice of extracting 13 **MFCC** aligns with standard speech processing practices. Increasing this number (e.g., extracting 20 or more coefficients) would include higher-order spectral details, but these are often less relevant for speech-related tasks and could introduce unnecessary noise, making classification or recognition tasks less efficient. Conversely, selecting fewer coefficients (e.g., 5 or 6) would discard essential spectral features, reducing the ability to characterize the vocal signal effectively.

Once the MFCC matrix has been calculated for each speech segment, a centroid was calculated by averaging along the columns of this matrix. The centroids were divided into two sets: training set and test set. The centroids contained in the training set simulate the acquisitions made in the enrollment phase, and then saved in the database, with which to make the comparison during the Identification Phase (Def. V). On the other hand, the centroids contained in the test set will be compared with those of the training set using the Euclidean distance as a metric. The system will associate the centroid with the user showing the lowest Euclidean mean distance and below a fixed threshold specified by the system.

The fixed threshold defines the maximum acceptable Euclidean mean distance between some user's centroids saved in database (○) and new centroid (△). If the distance is within this limit, the system associates the centroid with the closest user; otherwise, it results in a "No Match".

The threshold was determined empirically through trial and error, as there is no universally optimal value. A lower threshold reduces false positives but may increase false negatives, while a higher threshold has the opposite effect. Finding the right balance is crucial for optimizing the system's accuracy and reliability.

Since the representation of centroids is difficult due to the high dimensionality, PCA was applied on the centroid matrix to transform the data into a space of lower dimensionality making it possible to represent it in a system with only two dimensions. The representation was not always effective due to the excessive reduction of dimensionality.

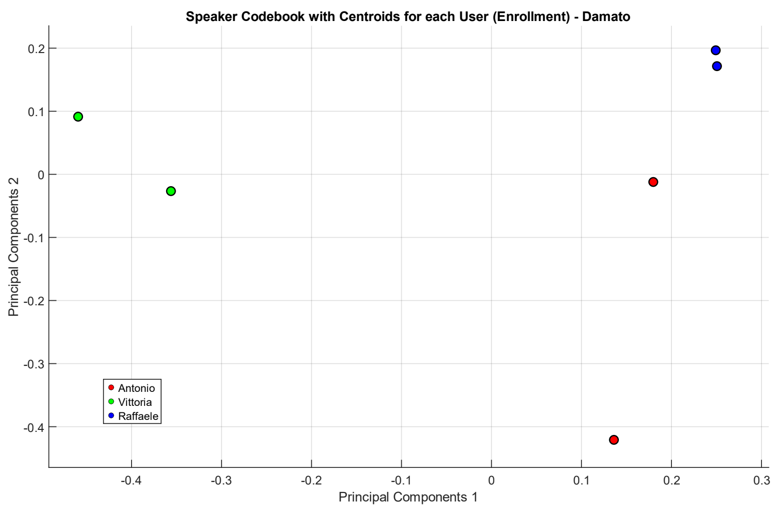
In the figures below, all the centroids are represented, reduced to two dimensions thanks to the use of PCA, associated with the different speakers and the different phases they represent. The different speakers are represented through the use of the following colors:

- Red → Antonio;
- Green → Vittoria;
- Blue → Raffaele.

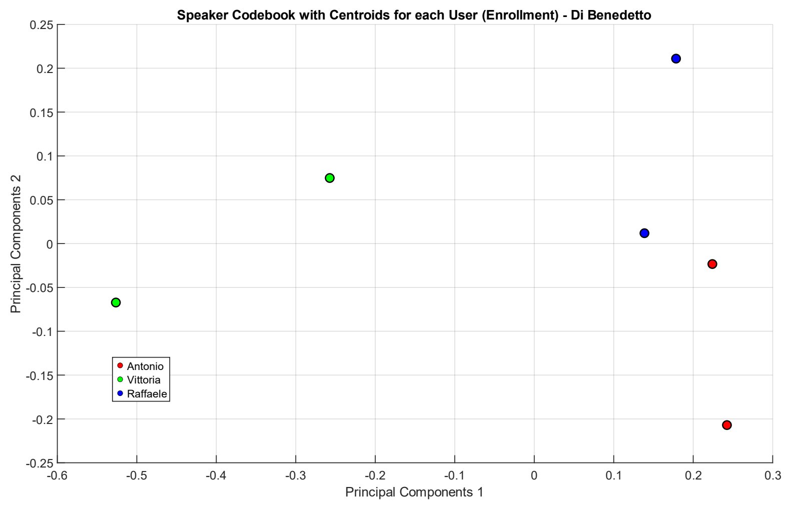
While the phase represented by each centroid is classified through the use of the following shapes:

- Circle (○) → centroids calculated during the enrollment phase;
- Triangle (△) → centroids calculated during the identification phase.

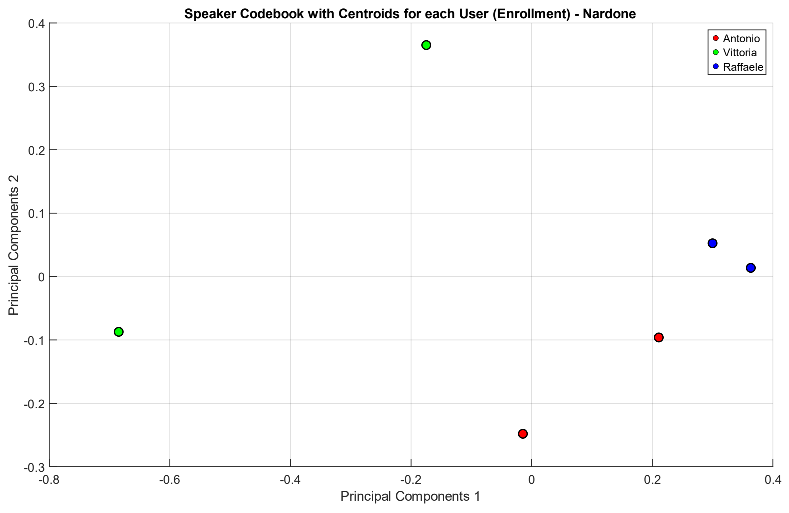
There is also another category labeled as "No Match" in white, exclusive to triangular centroids, not assigned to a specific speaker.



(a)

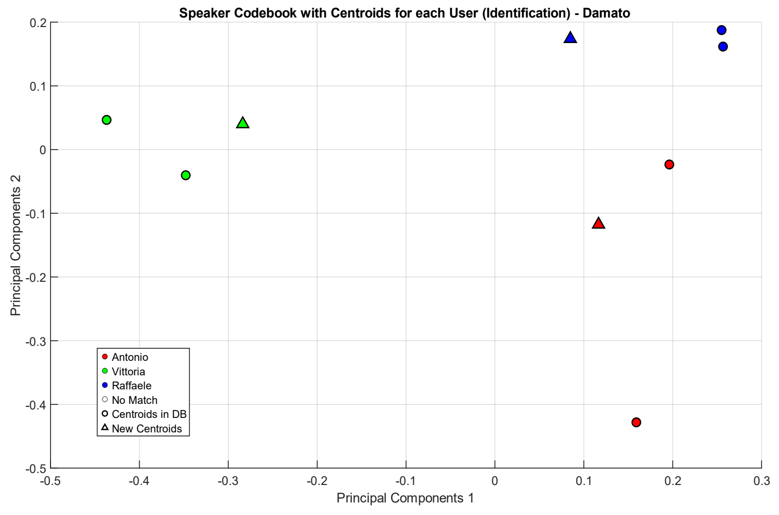


(b)

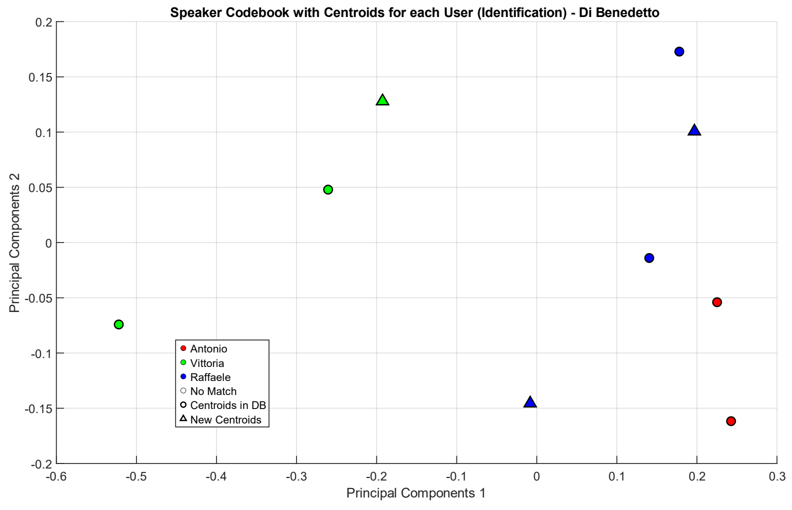


(c)

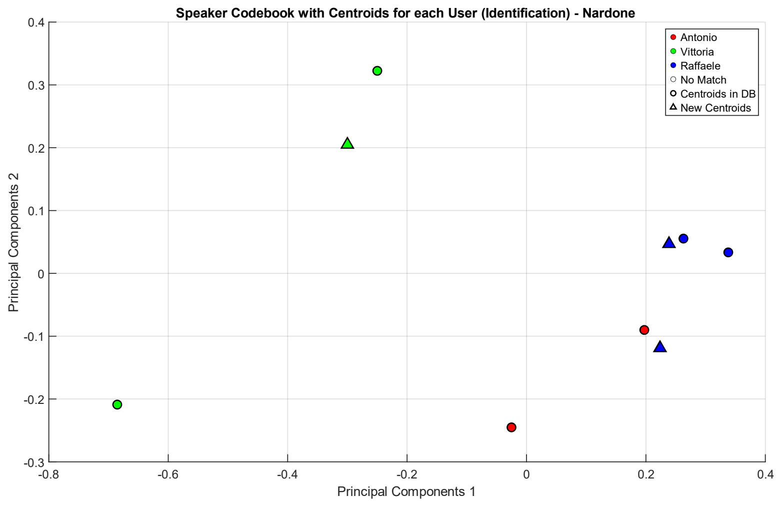
Figure 16: Principal Component Analysis And Centroids Definition (Enrollment Phase)



(a)



(b)



(c)

Figure 17: Principal Component Analysis And Centroids Definition (Identification Phase)

From Table 1 analysis shows that the speaker recognition system has just some issues in the classification of users, with variable results depending on the surname pronounced.

"DAMATO"	**Antonio**	**Vittoria**	**Raffaele**	**No Match**
Antonio	✓			
Vittoria		✓		
Raffaele			✓	

(a)

"NARDONE"	**Antonio**	**Vittoria**	**Raffaele**	**No Match**
Antonio			✗	
Vittoria		✓		
Raffaele			✓	

(b)

"DI BENEDETTO"	**Antonio**	**Vittoria**	**Raffaele**	**No Match**
Antonio			✗	
Vittoria		✓		
Raffaele			✓	

(c)

Table 1: Speaker Recognition Results For The Surnames 'Damato', 'Nardone' And 'Di Benedetto'

We observe that:

- In *Table 1(a)*, related to the surname "DAMATO", the system showed ideal behavior, correctly recognizing all speakers without classification errors. This leads to a 100% recognition rate, demonstrating that for this specific condition the model is able to effectively distinguish between different users.
- In *Table 1(b)*, which concerns the surname "NARDONE", the system showed a slight degradation of performance. While Victoria and Raphael were correctly identified, Anthony was misclassified as Raphael. This error indicates an overlap in Antonio and Raffaele's vocal characteristics for this specific word, suggesting that the system may have difficulty separating their voices under certain conditions.
- In *Table 1(c)*, which analyzes the surname "DI BENEDETTO", the pattern observed in the previous case is repeated. Once again, Raffaele and Vittoria were correctly recognized, but Antonio was mistakenly attributed to Raffaele. This error, which occurs in two of the three cases examined, highlights a tendency for the system to confuse Anthony with Raphael, suggesting a possible similarity in their vocal patterns or a weakness in the discrimination between these two specific voices.

We observe that system performance is optimal for some cases, while in others, specific errors emerge that appear to follow a consistent pattern. This recurrence of the error suggests that the main cause is not a systematic problem in recognition, but a difference in vocal timbre between the speakers. In particular, the fact that the woman and a man are always correctly identified, while the two men are confused with each other, can be attributed to a greater tonal similarity between Antonio and Raffaele. This suggests that the system may have difficulty separating male voices with similar acoustic characteristics, while it can more easily distinguish voices with more marked timbral differences, such as those of a man and a woman. Probably, with a higher sampling rate and a more complex noise filtering system than the one adopted in our solution, the system could have achieved a better recognition rate by allowing finer details to be captured in voices, improving discrimination between speakers with similar timbres.

An important role in the performance of the system was played by the use or not of the normalization technique, the application of which depends on the context of analysis considered, since each method affects the distribution of data and their interpretation in a different way. The scarcity of the data acquired did not allow to carry out an analysis on a wider range of values and therefore obtain a more precise estimate of the actual performance that could be obtained in a real context.

We obtained these results after carrying out various tests with and without normalization:

- When no type of normalization was employed, the obtained accuracy of the recognition system was 44.4% (4 correct recognitions out of 9);
- When Z-Score normalization was used, the accuracy of the system was 77.7% (7 correct recognitions out of 9);
- With Min-Max normalization, the overall accuracy of the system was 55.5% (5 correct recognitions of out of 9).

	Damato	**Nardone**	**Di Benedetto**	**Score**
No Normalization	66.6%	33.3%	33.3%	44.4%
Z-Score	100%	66.6%	66.6%	77.7%
Min-Max	66.6%	66.6%	33.3%	55.5%

Table 2: Accuracy Score As The Normalization Technique Used Changes

The *Table 2* highlights the impact of different normalization techniques on performance evaluation. Each experimented method significantly alters the relative scores, demonstrating how normalization can influence comparisons and rankings.

- **No Normalization** presents raw percentages, which may not be ideal when values are on different scales.
- **Z-Score Normalization** standardizes the data based on mean and standard deviation, amplifying differences and leading to the highest performance variations (e.g., Damato reaching 100%). This method it amplifies the differences and allows you to better highlight the relative variations between the elements: this approach is suitable in the case where the data follow a normal distribution, i.e. which therefore has few outlier values, while it can exaggerate the disparities, as observed in the case of "Damato" who reaches 100%.
- **Min-Max Normalization** rescales values to a fixed range, maintaining relative differences while reducing extreme variations. This approach provides a more balanced adjustment but may not be as effective if outliers are present.

In our case, the Z-Score gave the most promising results.

5 CONCLUDING NOTES AND POSSIBLE OBSERVATIONS

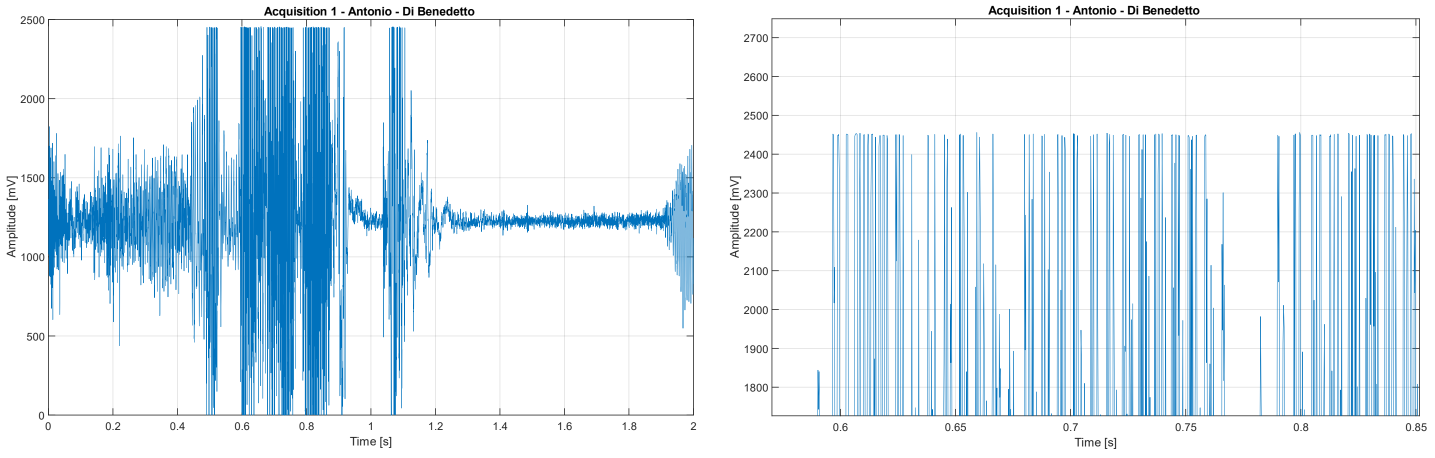
5.1 Observations

As for the audio capture with the microphone, it had to be replaced. When capturing with the first microphone provided, an issue of high signal levels was found even when there was only ambient noise. Using a second microphone, it was ascertained that the problem stemmed from a malfunction of the first, which made the data acquired for analysis unusable.

Finally, a further problem was found in sampling at high frequencies. The microcontroller could not keep up with serial printing of real-time data due to the time required for *the pritnf* operation, causing delays and data loss. To solve this limitation, it was decided to buffer the data, which allowed the data to be temporarily accumulated before sending it in chunks, allowing a sampling rate of 4 kHz to be used. The further limitation encountered, following this method of data acquisition and printing, was the size of the card's memory.

Anyway, the problem for the low sampling rate has been further discussed with its own solution in the previous chapter, allowing us to achieve better results with regard to the quality of the acquired vocal signals.

Another problem encountered was microphone saturation. The latter generates a weak signal that passes through a MAX9814 amplifier that has a DC offset (MIC_{OUT} Bias) of 1.23V (which represents the "rest" value of the microphone), around which the signal will oscillate. The amplifier is designed to have a maximum excursion of around 2.4 V_{PP} and a maximum output voltage of about 2.45V (MIC_{OUT} High Output Voltage). Having the DC offset, even without using an excessively high tone of voice, the output voltage quickly reaches the value at which the microphone saturates, therefore about 2.45V, not allowing to obtain the complete voice signal and therefore losing some spectral components useful for a possible recognition. A possible solution to solve this problem is proposed within the amplifier datasheet and involves the introduction of a 1μF C_{OUT} capacitor, which would eliminate the direct current offset of 1.23V preventing saturation, since the signal would not easily exceed the 2.45V threshold.



(a) (b)

Figure 18: (a) Example Of Saturated Signal; (b) Focus On Saturation At 2450 mV

5.2 Conclusion

The laboratory experience has allowed us to deepen the process of acquisition, processing and extraction of features from physiological signals. The activity highlighted the importance of proper signal acquisition, as data quality directly affects the subsequent processing and analysis steps.

The experimental activities carried out allowed to analyze some biometric methodologies for speaker recognition, highlighting the main challenges related to their implementation and data acquisition.

The system showed a good level of accuracy, but was strongly influenced by the quality of the microphone and the presence of ambient noise. The use of pre-processing and feature extraction techniques (such as MFCC) has proved to be fundamental to improve the distinction between speakers. However, further optimizations, such as the use of more advanced machine learning models, could further improve the reliability of the system.

In general, the results obtained confirm that biometric recognition is a technology with high application potential, but that it requires special attention in the signal acquisition phase and in data processing to obtain reliable performance. Improved filtering techniques, optimization of acquisition parameters and integration with machine learning models could be future developments to make these systems more robust and efficient.