STUDY AND EVALUATION OF BIOMETRIC TECHNIQUES: GAIT RECOGNITION

Anna Vittoria Damato
Antonio Nardone
Carolina Sbiroli
Raffaele Di Benedetto

Academic Year 2024/2025

Electronic systems for Biometrics and biosensing

Table Of Contents

Table Of Figures

Table Of Tables

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
Wi-Fi – Wireless Fidelity
MEMS – Micro Electro Mechanical Systems
I²C – Inter Integrated Circuit
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
WS – Wearable Sensor

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. **Gait Recognition**: technique that is based on the analysis of a person's walk. Using cameras and image processing algorithms, the system detects the silhouette of the subject and extracts its dynamic characteristics. These parameters are then compared with those in a database to identify the individual.

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.

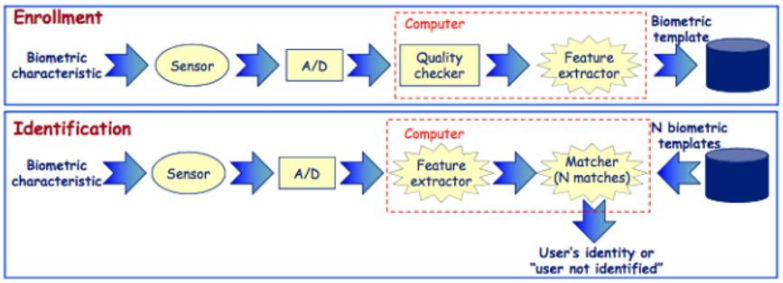

Figure 1: Outline of the identification process in a biometric system

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.

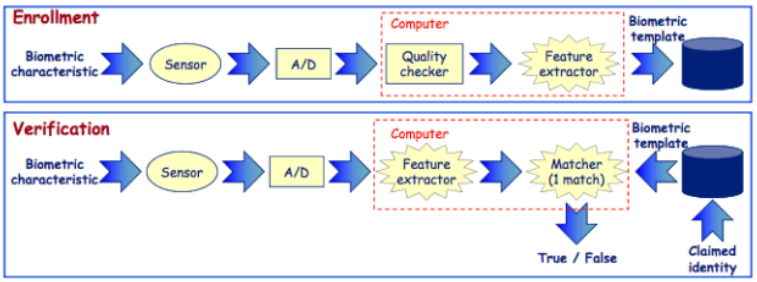

Figure 2: Outline of the authentication process in a biometric system

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. **Gait Cycle:** single gait cycle during which data acquisition is carried out. Each gait cycle has two main phases:

- Stance Phase, the phase during which the foot remains in contact with the ground;
- Swing Phase, the phase during which the foot is not in contact with the ground.

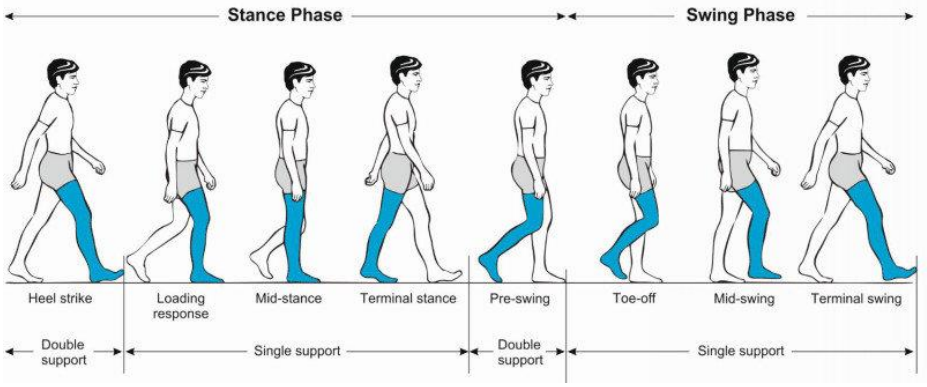

Figure 3: Gait Cycle Divided Into Stance Phase And Swing Phase

XIII. **WS-based Gait Recognition:** In WS-based Gait Recognition, gait is collected using body-worn motion recording (MRI) sensors. MRI sensors can be worn in different places on the human body. The gait acceleration, which is recorded by the MRI sensors, is used for authentication via gait cycle analysis. One of the main advantages of this type of gait recognition is the use of non-invasive acquisition techniques (unobtrusive).

XIV. **MEMS Accelerometer:** a sensor capable of detecting the acceleration along one or more axes of an object under test, within an inertial reference frame. It consists of a test mass suspended by the use of anchored mechanical suspensions.

One of the most widely used test mass displacement detection mechanisms is capacitive. In capacitive accelerometers, a differential change in capacitance (between the test mass and the frame) is usually detected. The movement of the test mass causes some capacities to increase and others to decrease.

The basic configurations are either open loop or closed loop. Of particular interest to us is the open-loop one, which is an operating configuration of inertial sensors in which the system directly measures the displacement of the test mass without applying active feedback to correct it. In this configuration, the signal generated by the sensing mechanism is read and used to estimate the force or acceleration without interfering with the movement of the test mass.

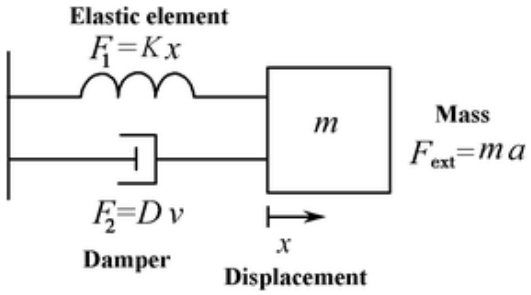

Figure 4: Mechanical Model Of A MEMS Accelerometer

1 OBJECTIVES OF THE REPORT

The purpose of this report is to document the activities carried out during the laboratory sessions, in which different recognition techniques based on the processing of electronic signals and biometric data were tested.

The main objectives were structured as follows:

- **Gait signal acquisition:** acquire signals through a triaxial accelerometer capable of recording acceleration variations along the three spatial axes, positioning the sensor on the front and side of the leg.
- **Application of pre-processing techniques:** process acquired signals using filtering and normalization techniques and analyze swing and stance phase using plots.
- **Feature extraction:** analysis of the extracted features from identified swing and stance phases.

2 HARDWARE

2.1 Components

2.1.1 STM32 Nucleo F091RC

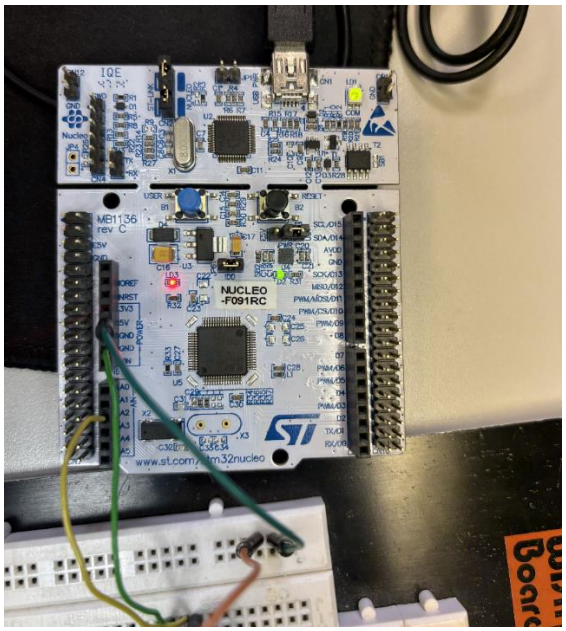

Figure 5: STM32 NUCLEO F091RC

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 STM X-NUCLEO-IKS01A2

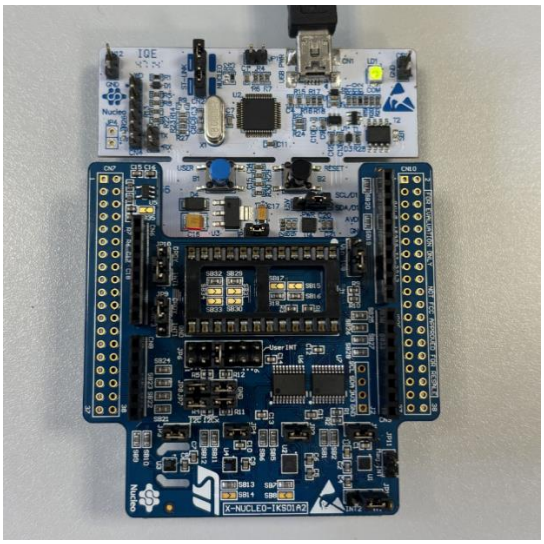

Figure 6: STM X-NUCLEO-IKS01A2 Accelerometer

The X-NUCLEO-IKS01A2 is a sensory expansion board designed to be used with the STM32 Nucleo platforms. This module enables the acquisition of environmental and motion data through the integration of different MEMS sensors. These include the LSM6DSL, a three-axis accelerometer and gyroscope with selectable range, which allows you to acquire movement and orientation data. In addition to the latter, and many others, the board has a DIL24 socket that allows expansion with additional compatible MEMS sensors.

The X-NUCLEO-IKS01A2 is designed to operate under standard environmental and integration conditions for embedded devices. It works with a supply voltage of 3.3V or 5V, depending on the configuration of the Nucleo motherboard to which it is connected. Communication with the microcontroller takes place via an I²C interface, which supports multiple addressing configurations for flexible management of data from the sensors. Power consumption varies depending on the sensors activated and the operating mode selected, allowing optimizations for low-power applications.

The operation of the X-NUCLEO-IKS01A2 is based on the transmission of data in real time through the I²C interface. The STM32 microcontroller receives information from the built-in sensors, in particular, the LSM6DSL sensor acts as a hub for I²C data acquisition, allowing information to be collected from multiple sources and reducing the computational load of the main microcontroller.

2.2 Circuits

2.2.1 Accelerometer

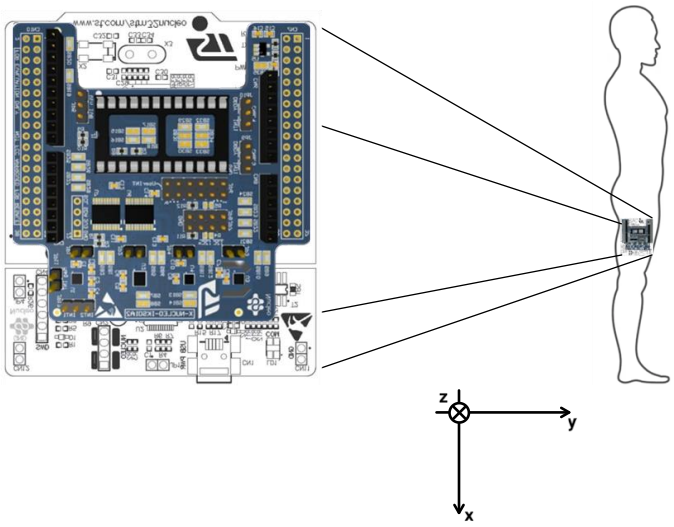

Figure 7: Circuit Diagram For Gait Acquisition With Sensor Positioned On The Side Of The Leg With Axes Orientation

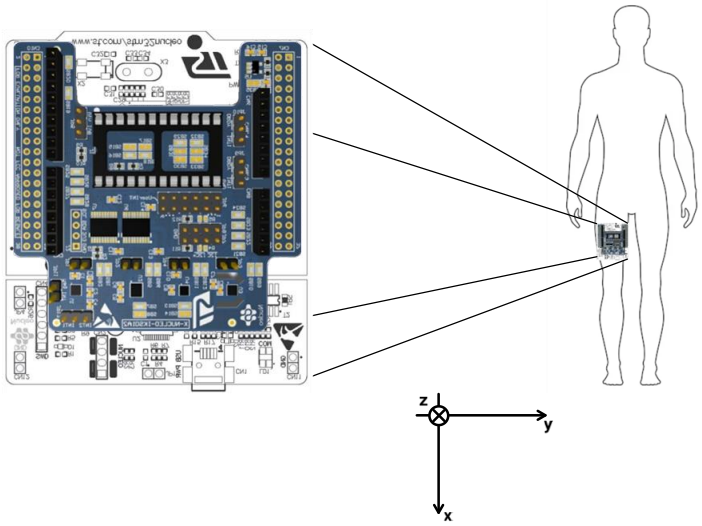

Figure 8: Circuit Diagram For Gait Acquisition With Sensor Positioned On The Top Of The Leg With Axes Orientation

To acquire acceleration data on three axes, an expansion board with a triaxial accelerometer was used, connected to the microcontroller. In this configuration, the only physical connection was made between the microcontroller and the computer via a USB-USB mini B cable.

- **Communication and power supply**:
  - The USB cable provides both power to the microcontroller and the communication channel for data transmission between the microcontroller and the computer.
  - The microcontroller, in turn, manages the accelerometer board via the intended communication protocol (I2C).

The microcontroller equipped with an expansion board has been positioned both on the side of the leg (Figure 7) and on the top (Figure 8).

The orientation and the axes used for an accelerometer vary depending on whether it is positioned on the top or the side of the leg. When the accelerometer is placed on the top of the leg, the x-axis points downward toward the ground, the y-axis is directed inward, toward the inner part of the leg, and the z-axis enters into the thigh.

However, when the accelerometer is positioned on the side of the leg, the y-axis points outward, away from the thigh, the x-axis is directed downward, and the z-axis still enters into the thigh. This difference in orientation must be taken into account when analyzing data, as the reference system changes accordingly.

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 Gait Acquisition

```
#include "mbed.h"
#include <cstdint>
#include <cstdio>

// Sensor I2C address
#define SENSOR_ADDRESS 0xD6

// Register addresses
#define WHO_AM_I 0x0F
#define CTRL1_XL 0x10
#define POWER_DOWN 0x0000
#define ODR_12Hz 0b00010000 // Output data rate 12Hz
#define FS_2G 0b00000000    // ±2g range
#define FS_4G 0b00010000    // ±4g range
#define FS_8G 0b00011000    // ±8g range
#define FS_16G 0b00000100   // ±16g range

// Sensitivity values (mg/LSB)
#define SENS_2G 0.061
#define SENS_4G 0.122
#define SENS_8G 0.244
#define SENS_16G 0.732

// Output registers for acceleration
#define OUTX_L_XL 0x28
#define OUTX_H_XL 0x29
#define OUTY_L_XL 0x2A
#define OUTY_H_XL 0x2B
#define OUTZ_L_XL 0x2C
#define OUTZ_H_XL 0x2D

// Output registers for temperature (unused)
#define OUT_TEMP_L 0x20
#define OUT_TEMP_H 0x21

// Terminal clear command
#define CLEAR "\f\033[?25I"

// Initialize I2C communication
I2C i2c(I2C_SDA, I2C_SCL);

// Function to read a register from the accelerometer
char read_reg(char reg_address) {
    char data_write[1];
    char data_read[1];
    data_write[0] = reg_address;

    // Send register address and read back value
    i2c.write(SENSOR_ADDRESS, data_write, 1, 0);
    i2c.read(SENSOR_ADDRESS, data_read, 1, 0);

    return data_read[0];
}

// Function to write a value to a register
void write_reg(char reg_address, char data) {
    char data_write[2];
    data_write[0] = reg_address;
    data_write[1] = data;

    // Send register address and data
    i2c.write(SENSOR_ADDRESS, data_write, 2, 0);
}
```

```
int main() {
    // Clear terminal and print a new line
    printf(CLEAR "\n\r");

    char value;
    short buf[4];
    float a = SENS_2G, x = 0, y = 0, z = 0;

    // Read "Who am I?" register to check sensor identity
    value = read_reg(WHO_AM_I);
    printf("Who am I? %X \n\r", value);

    // Configure accelerometer: 12Hz output rate, ±2g range
    write_reg(CTRL1_XL, ODR_12Hz | FS_2G);

    // Start timer to limit execution to 10 seconds
    Timer timer;
    timer.start();

    // Read accelerometer data for 10 seconds
    while (timer.read() < 10.0) {
        // Read acceleration X-axis
        buf[0] = read_reg(OUTX_H_XL);
        buf[0] <<= 8;
        buf[0] |= read_reg(OUTX_L_XL);

        // Read acceleration Y-axis
        buf[1] = read_reg(OUTY_H_XL);
        buf[1] <<= 8;
        buf[1] |= read_reg(OUTY_L_XL);

        // Read acceleration Z-axis
        buf[2] = read_reg(OUTZ_H_XL);
        buf[2] <<= 8;
        buf[2] |= read_reg(OUTZ_L_XL);

        // Convert raw data to acceleration values in mg
        x = buf[0] * a;
        y = buf[1] * a;
        z = buf[2] * a;

        // Print acceleration data
        printf(" ACC(xyz):%.1f %.1f %.1f \n\r", x, y, z);
    }

    // Print end message
    printf("-----------------------End of Walk --------------------------------");
```

Figure 9: Firmware For Reading Triaxial Acceleration Data With Accelerometer

The code uses the Mbed library to acquire acceleration data from a triaxial accelerometer connected via I2C, collecting information about movement while walking. The sensor address is defined using the *constant SENSOR_ADDRESS (0xD6)* and key registers are set for configuring and reading data, including *WHO_AM_I* for sensor identification and *CTRL1_XL* for accelerometer configuration. Several configuration parameters are defined such as refresh rate (*ODR_12Hz*), sensitivity ranges *(FS_2G*, *FS_4G*, etc.) and related conversion scales (*SENS_2G*, SENS_4G, etc.). Acceleration output registers along the X, Y, and Z axes are also specified.

The *i2c* object is instantiated for communication with the sensor. The *read_reg* function allows you to read the value of a register by sending its address and receiving the corresponding data. The *write_reg function* writes a value to a sensor register by sending the address first and then the data to be written.

At the beginning of the *main* function, a terminal cleaning command (*CLEAR)* is printed. A value variable is then declared, a *buf* array to store the raw accelerometer data and some *variables a, x, y, z* to manage the accelerometer values. Communication with the sensor is then initiated by reading the *WHO_AM_I* register to verify its identity, printing the read value. The accelerometer is then configured by writing a value in the *register CTRL1_XL* that enables operation at 12 Hz with a range of ±2g.

We chose a gait signal acquisition frequency of 12 Hz, considering it adequate compared to higher frequencies for several reasons. First, human walking is characterized by low-frequency components, typically between 0 and 5 Hz (typically 40-50 steps per minute). The use of a frequency of 12 Hz therefore makes it possible to comply with the Nyquist criterion, avoiding aliasing and at the same time ensuring efficient acquisition with less memory and computational resource consumption than higher frequencies such as 50 Hz or 100 Hz, which would be redundant for the type of signal being analyzed.

A *Timer* is started to limit data capture to 10 seconds. In the *while* loop, as long as the elapsed time is less than 10 seconds, the output registers for the three axes of the accelerometer are read. Each acceleration value consists of two bytes (high and low), which are combined to obtain the complete 16-bit data. Finally, acceleration values along the X, Y, and Z axes are printed on the console. When the cycle is complete, a message is printed indicating the end of data acquisition.

The acquisition lasted 10 seconds to ensure sufficient data collection to accurately represent the gait cycle. Considering that an average subject performs about 1-1.5 steps per second, in 10 seconds it is possible to collect at least 10-15 cycles of the gait, an adequate amount for statistical analysis and to ensure good robustness of the results and allow the user to walk the defined path.

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.

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 Gait Analysis - TOP

```
clc;
clear all;
close all;

% Main settings
utenti = {'antonio', 'vittoria', 'carolina', 'raffaele'}; % User names
posizioni = {'top'}; % Sensor positions
fase_labels = {'Stance', 'Swing'}; % Gait cycle phases
num_campioni = 279; % Number of samples to take per column
fs = 27.9; % Sampling frequency (279 samples in 10 seconds)

% Function to load data from a file
function dati = carica_dati(nome_file, num_campioni)
    % Read the file as a table of strings
    dati_raw = readtable(nome_file, 'ReadVariableNames', false, 'TextType', 'string');

    % Number of rows in the file
    num_righe = height(dati_raw);

    % Find the first row containing valid data
    first_data_row = find(contains(dati_raw{:, 1}, 'ACC(xyz):'), 1);

    % If no valid data is found, alert the user
    if isempty(first_data_row)
        error('No valid data found in the file.');
    end

    % Slice the table to remove header rows
    dati_raw = dati_raw(first_data_row:end, :);

    % Check if the file contains enough rows for the required number of samples
    if height(dati_raw) < num_campioni
        warning('The file contains fewer rows than required for %d samples. Using available
data.', num_campioni);
        num_campioni = height(dati_raw);  % Adjust the number of samples
    end

    % Initialize a matrix for numeric data (9 columns: 3 per acquisition)
    dati = zeros(num_campioni, 9);   % Each row contains 9 values: 3 for each of the 3
acquisitions

    % Loop to extract data from each row of the file
    for i = 1:num_campioni
        % Extract data from row i
        riga = dati_raw{i, :};

        % Loop through each acquisition (3 columns per row)
        for col_idx = 1:3
            % Extract XYZ values for the corresponding column
            valori = str2double(regexp(riga{col_idx}, '-?\d+(\.\d+)?', 'match'));  % Extract
numbers

            % Check if there are exactly 3 values (X, Y, Z)
            if length(valori) == 3
                % Assign the values to the 3 appropriate columns
                dati(i, (col_idx - 1) * 3 + 1) = valori(1);  % X-axis
                dati(i, (col_idx - 1) * 3 + 2) = valori(2);  % Y-axis
                dati(i, (col_idx - 1) * 3 + 3) = valori(3);  % Z-axis
            else
                warning('Row %d, column %d does not contain 3 valid numbers for X, Y, Z.', i,
col_idx);
            end
        end
    end
end
```

```
% Signal filtering function
function segn_filtrato = filtra_segnale(segnale, fs)
    fc = 3; % Cutoff frequency
    [b, a] = butter(4, fc / (fs / 2), 'low'); % Low-pass filter
    segn_filtrato = filtfilt(b, a, segnale); % Apply filter
end

% Z-Score standardization function
function segn_standardizzato = standardizza_segnale(segnale)
    segn_standardizzato = zeros(size(segnale)); % Pre-allocate matrix
    for col = 1:size(segnale, 2) % Iterate through axes (columns)
        media = mean(segnale(:, col)); % Mean
        deviazione_std = std(segnale(:, col)); % Standard deviation
        segn_standardizzato(:, col) = (segnale(:, col) - media) / deviazione_std; % Z-Score
standardization
    end
end

% Function to find peaks and valleys to determine gait cycle intervals (only specific axis)
function [swing_intervals, stance_intervals] = trova_intervalli_gait_cycle(dati_normalizzati,
fs)
    % Initialize variables to store intervals
    swing_intervals = [];
    stance_intervals = [];

    % Find peaks in the signal
    [picchi, pos_picchi] = findpeaks(dati_normalizzati, 'MinPeakDistance', 20, 'MinPeakHeight',
1.5);

    % Find valleys in the signal
    [minimi, pos_minimi] = findpeaks(-dati_normalizzati);  % Valleys are peaks in the inverted
signal

    % Loop through each peak found
    for i = 1:length(picchi)
        picco_idx = pos_picchi(i);  % Peak index

        % Find the first minimum before and after the peak
        min_prim_before = find(pos_minimi < picco_idx, 1, 'last');
        min_prim_after = find(pos_minimi > picco_idx, 1, 'first');

        if ~isempty(min_prim_before) && ~isempty(min_prim_after)
            % Define the Swing interval
            swing_intervals       =       [swing_intervals;       pos_minimi(min_prim_before),
pos_minimi(min_prim_after)];

            % Find the first minimum after peak i and the first minimum before the next peak
            if i + 1 <= length(picchi)
                min_after_i = find(pos_minimi > pos_picchi(i), 1, 'first');
                min_before_next_peak = find(pos_minimi < pos_picchi(i+1), 1, 'last');

                if ~isempty(min_after_i) && ~isempty(min_before_next_peak)
                    % Define the Stance interval
                    stance_intervals     =     [stance_intervals;     pos_minimi(min_after_i),
pos_minimi(min_before_next_peak)];
                end
            end
        end
    end
end

% Function to extract swing and stance phase features
function  estrai_feature_swing_stance(swing_intervals,  stance_intervals,  dati_z,  fs,  utente,
acquisizione)
    % Select samples that belong to the swing and stance ranges
    swing_samples = [];
    stance_samples = [];

    % Add Swing Phase Samples
    for i = 1:size(swing_intervals, 1)
        swing_samples = [swing_samples; dati_z(swing_intervals(i, 1):swing_intervals(i, 2))];
    end
```

```
% Add Stance Stage Samples
    for i = 1:size(stance_intervals, 1)
        stance_samples  =  [stance_samples;  dati_z(stance_intervals(i,  1):stance_intervals(i,
2))];
    end

    % Merge Swing and Stance Champions
    all_samples = [swing_samples; stance_samples];

    % Average acceleration for selected samples only
    mean_acc = mean(all_samples) / 100;

    % Calculate the standard deviation of acceleration for selected samples
    std_acc = std(all_samples) / 100;

    % Duration of the Swing phase
    swing_durations = (swing_intervals(:,2) - swing_intervals(:,1)) / fs;
    mean_swing_duration = mean(swing_durations);

     % Duration of the Stance phase
    stance_durations = (stance_intervals(:,2) - stance_intervals(:,1)) / fs;
    mean_stance_duration = mean(stance_durations);

    % Cadence calculation (steps per minute)
    num_steps = length(swing_intervals);
    total_time = length(dati_z) / fs;
    cadence = (num_steps / total_time) * 60;

    % Calculation of ratios
    swing_stance_ratio = mean_swing_duration / mean_stance_duration;
    swing_gait_ratio = mean_swing_duration / (mean_swing_duration + mean_stance_duration);

    % Feature printing
    fprintf('Utente: %s | Acquisizione: %d\n', utente, acquisizione);
    fprintf('Mean Acceleration: %.4f m/s^2\n', mean_acc);
    fprintf('Std Acceleration: %.4f m/s^2\n', std_acc);
    fprintf('Swing Duration: %.4f s\n', mean_swing_duration);
    fprintf('Stance Duration: %.4f s\n', mean_stance_duration);
    fprintf('Cadence: %.2f steps/min\n', cadence);
    fprintf('Swing/Stance Ratio: %.4f\n', swing_stance_ratio);
    fprintf('Swing/Gait Ratio: %.4f\n', swing_gait_ratio);
    fprintf('---------------------------------------\n');
end

% Main loop to add gait cycle interval calculations (only Z-axis)
for i = 1:length(utenti)
    utente = utenti{i};
    for j = 1:length(posizioni)
        posizione = posizioni{j};
        % File name
        nome_file = sprintf('gait_%s_%s.xlsx', utente, posizione);

        % Load data
        dati = carica_dati(nome_file, num_campioni);

        % Check data validity
        if isempty(dati)
            warning('Data for %s were not loaded correctly', nome_file);
            continue;
        end

        % Preprocessing
        dati_filtrati = filtra_segnale(dati, fs); % Low-pass filtering
        dati_normalizzati = standardizza_segnale(dati_filtrati); % Z-Score standardization

        % Loop through each acquisition (3 columns at a time)
        for k = 1:3:9
            % Calculate Swing and Stance intervals (only Z-axis of the current column)
            [swing_intervals,                        stance_intervals]                        =
trova_intervalli_gait_cycle(dati_normalizzati(:, k+2), fs);

            utente_capitalized = strcat(upper(utente(1)), lower(utente(2:end)));
```

```
            % Plot the Z-axis signal (normalized)
            figure;
            t = (0:size(dati_normalizzati, 1) - 1) / fs; % Time axis based on sampling frequency

            % Plot the normalized Z-axis signal (black)
            plot(t, dati_normalizzati(:, k+2), 'k-', 'LineWidth', 1.5);
            hold on;

           % Color Y signal only in uncolored segments
            z_plot = plot(nan, nan, 'k', 'LineWidth', 1.5);

            % Color Swing Phase
            swing_plot = plot(nan, nan, 'g', 'LineWidth', 2);

           % Color Swing intervals in green
            for m = 1:size(swing_intervals, 1)
                t_swing = t(swing_intervals(m, 1):swing_intervals(m, 2));
                segn_swing=dati_normalizzati(swing_intervals(m,   1):swing_intervals(m,   2),
                k+2);
                plot(t_swing, segn_swing, 'g', 'LineWidth', 2); % Green color for Swing
            end

            % Color Stance Phase
            stance_plot = plot(nan, nan, 'r', 'LineWidth', 2);

             % Color Stance intervals in red
            for m = 1:size(stance_intervals, 1)
                t_stance = t(stance_intervals(m, 1):stance_intervals(m, 2));
                segn_stance = dati_normalizzati(stance_intervals(m, 1):stance_intervals(m, 2),
k+2);
                plot(t_stance, segn_stance, 'r', 'LineWidth', 2); % Red color for Stance
            end

            % Add vertical lines at peaks and valleys
            for m = 1:size(swing_intervals, 1)
                line([t(swing_intervals(m, 1)) t(swing_intervals(m, 1))], ylim, 'Color', 'k',
'LineStyle', '--', 'LineWidth', 1);
                line([t(swing_intervals(m, 2)) t(swing_intervals(m, 2))], ylim, 'Color', 'k',
'LineStyle', '--', 'LineWidth', 1);
            end
            for m = 1:size(stance_intervals, 1)
                line([t(stance_intervals(m, 1)) t(stance_intervals(m, 1))], ylim, 'Color', 'k',
'LineStyle', '--', 'LineWidth', 1);
                line([t(stance_intervals(m, 2)) t(stance_intervals(m, 2))], ylim, 'Color', 'k',
'LineStyle', '--', 'LineWidth', 1);
            end

            % Labels
            xlabel('Time [s]');
            ylabel('Acceleration [m/s^2]');
            % Add the legend
            legend([swing_plot, stance_plot, z_plot], ...
                {'Swing Phase', 'Stance Phase', 'Z-Signal'}, 'Location', 'best');
            grid on;

            sgtitle(sprintf('Normalized Signal: %s - %s (Acquisition %d)', utente_capitalized,
posizione, ceil(k/3))); % Title

            % Extract features for each cycle (only Z-axis)
            estrai_feature_swing_stance(swing_intervals,                    stance_intervals,
dati_normalizzati(:, k+2), fs, utente_capitalized, k);
        end
    end
end
```

Figure 10: Code For Gait Cycle Phases Recognition With Accelerometer Placed On The Top Of The Leg

The code presented is designed to analyze accelerometer data related to walking, focusing in

particular on the subdivision of the phases of the gait cycle, such as "Swing" and "Stance" (Def. XII), and on the extraction of different biomechanical characteristics from these phases. The first part of the code deals with loading and preparing the data. The accelerometer data is read from an Excel file that contains information about three axes (X, Y, Z) for each acquired sample. The loading is done in a tabular format, and the header rows are removed to get only the useful data. Next, the three axis information for each sample is extracted and assigned to an appropriately sized matrix.

Once the data has been loaded, the code applies the following steps:

- **Filtering.**
  A low-pass filter is used to remove any unwanted noise or fluctuations in the signals. This is done using a *butter* function that defines a Butterworth filter with a cutoff frequency of 3 Hz.
  This choice is motivated by the spectral analysis which showed that the most useful information in the gait signal is contained within this band, while higher frequencies are generally attributable to unwanted noise and vibrations. A higher cutoff filter, such as 10 Hz, may introduce unwanted noise, while a value below 2 Hz may excessively smooth the signal, risking losing important relative details that would allow to effectively distinguish the swing and stance phases.

- **Normalization.**
  After filtering, the data is normalized using a Z-Score normalization, which makes them comparable with each other. In gait acquisition, data may vary between subjects and there may be anomalies (irregular steps, physiological variations...), the impact of which can be reduced thanks to normalization. Z-Score Normalization is performed on each axis separately, subtracting the mean of the values and dividing by the standard deviation.

  Another normalization technique we could have used is Min-Max normalization, which transforms the data into a fixed range, usually between 0 and 1, while maintaining the original proportions between the values. This can be useful in some contexts, but presents critical issues when analyzing walking data. One of the main issues is sensitivity to outliers, which are extremely high or low values that can be due to acquisition errors or irregular steps. If a single particularly long or short step is recorded, it will affect the entire transformation, compressing the normal data into a confined space and reducing its significance. In addition, min-max normalization makes it difficult to compare different subjects, because by forcing all the data into a specific range, it can mask natural changes in pitch that might be relevant to the analysis.

  The choice fell on the Z-score because it has the advantage of being less influenced by outliers, since extreme values do not distort the entire dataset, and allows to maintain a more balanced distribution of information. In addition, it is particularly useful when comparing data from different subjects, because it allows them to be analyzed in terms of their natural variability, without forcing them into an arbitrary range.

- **Identification of the phases of the gait cycle.**
  The gait cycle is divided into two phases: "Swing" and "Stance". The algorithm looks for peaks and troughs in the accelerometer data to identify critical points in the gait cycle, such as the transition between the two phases. Peaks are used to identify the beginning of the "Swing" phase, while lows are used to determine the beginning of the "Stance" phase. Once the peaks and troughs have been identified, the time intervals in which each phase takes place are determined, allowing the two phases of the gait cycle to be clearly separated.

Once the ranges have been identified, the *estrai_feature_swing_stance function* calculates several metrics of interest. These include the average acceleration and its standard deviation on the Z-axis, the average duration of the Swing and Stance phases, the cadence (steps per minute), and the Swing/Stance ratio, which indicates the proportion of time spent in each phase compared to the complete cycle of the step.

- **Plotting of the Gait Cycle.**
  The code plots the normalized Z signal on a graph, highlighting the Swing and Stance ranges with different colors (green and red, respectively). Vertical lines are also drawn to mark the transitions between the two stages, making it clearer to visualize the transitions between step stages.
  Before plotting only the signal on the Z-axis, we also analyzed the other two axes to evaluate their possible use in this analysis. However, the most significant data was only on the Z-axis as it was the direction in which the user was moving. In fact, the Y-axis would have shown only the lateral movements of the user, which were not present as the path was straight. The X-axis, on the other hand, did not show any periodicity or pattern useful for recognizing the walk of different users.

- **Features Extraction.**
  After the graphical representation, the code extracts quantitative features from the data. It calculates mean and standard deviation of the acceleration, duration of the Stance and Swing phases, the cadence, the ratio between the duration of the Swing phase and the Stance phase and finally the ratio of the Swing phase to the entire step cycle.

3.2.2 Gait Analysis - SIDE

```
clc;
clear all;
close all;

% Set Default Parameters for Figures
%set(0, 'DefaultFigureColor', 'w'); % White background
set(0, 'DefaultAxesFontSize', 14); % Axis Font Size
%set(0, 'DefaultAxesFontWeight', 'Bold'); % Bold text
%set(0, 'DefaultLineLineWidth', 2); % Line thickness
set(0, 'DefaultAxesLineWidth', 1); % Axle thickness
%set(0, 'DefaultAxesGridLineStyle', '--'); % Dotted grid style
%set(0, 'DefaultAxesXGrid', 'on', 'DefaultAxesYGrid', 'on'); % Enable the grid
%set(0, 'DefaultFigurePosition', [200 200 800 600]); % Window size and position
set(0, 'DefaultLegendFontSize', 14);

% Main settings
users = {'antonio', 'vittoria', 'carolina', 'raffaele'}; % User names
positions = {'side'}; % Sensor positions
phase_labels = {'Stance', 'Swing'};
n_samples = 279; % Number of samples to take per column
fs = 27.9; % Sampling frequency (279 samples in 10 seconds)

% Function to load data from file
function data = load_data(file_name, n_samples)
    % Read the file as a table of strings
    raw_data = readtable(file_name, 'ReadVariableNames', false, 'TextType', 'string');

    % Get the number of rows in the file
    num_rows = height(raw_data);

    % Find the first row containing valid data
    first_data_row = find(contains(raw_data{:, 1}, 'ACC(xyz):'), 1);

    % If no valid data is found, raise an error
    if isempty(first_data_row)
        error('No valid data found in the file.');
    end

    % Slice the table to remove header rows
    raw_data = raw_data(first_data_row:end, :);

    % Check if the file contains at least n_samples rows
    if height(raw_data) < n_samples
        warning('The file contains fewer rows than required. Using available data.');
        n_samples = height(raw_data);
    end

    % Initialize a matrix for numerical data (9 columns: 3 per acquisition)
    data = zeros(n_samples, 9);

    % Loop to extract data from each row
    for i = 1:n_samples
        row = raw_data{i, :};

        % Loop for each acquisition (3 columns per row)
        for col_idx = 1:3
            % Extract XYZ values from the corresponding column
            values = str2double(regexp(row{col_idx}, '-?\d+(\.\d+)?', 'match'));

            % Check if exactly 3 values (X, Y, Z) are present
            if length(values) == 3
                data(i, (col_idx - 1) * 3 + 1) = values(1); % X axis
                data(i, (col_idx - 1) * 3 + 2) = values(2); % Y axis
                data(i, (col_idx - 1) * 3 + 3) = values(3); % Z axis
            else
                warning('Row %d, column %d does not contain 3 valid numbers.', i, col_idx);
            end
        end
    end
end
```

```
% Function to filter the signal
function filtered_signal = filter_signal(signal, fs)
    fc = 3; % Cutoff frequency
    [b, a] = butter(4, fc / (fs / 2), 'low'); % Low-pass filter
    filtered_signal = filtfilt(b, a, signal); % Apply filter
end

% Function to standardize the signal using Z-score
function standardized_signal = standardize_signal(signal)
    standardized_signal = zeros(size(signal));
    for col = 1:size(signal, 2)
        mean_val = mean(signal(:, col));
        std_dev = std(signal(:, col));
        standardized_signal(:, col) = (signal(:, col) - mean_val) / std_dev;
    end
end

% Main processing loop
for i = 1:length(users)
    user = users{i};
    for j = 1:length(positions)
        position = positions{j};
        file_name = sprintf('gait_%s_%s.xlsx', user, position);

        % Load data
        data = load_data(file_name, n_samples);

        % Check data validity
        if isempty(data)
            warning('Data for %s was not loaded correctly.', file_name);
            continue;
        end

        % Preprocessing
        filtered_data = filter_signal(data, fs); % Low-pass filter at 3 Hz
        normalized_data = standardize_signal(filtered_data); % Z-score standardization

        num_acquisitions = 3;
        for k = 1:num_acquisitions
            user_capitalized = strcat(upper(user(1)), lower(user(2:end)));

            % Create a new figure for each acquisition
            figure;
            sgtitle(sprintf('Normalized Signal: %s - %s (Acquisition %d)', user_capitalized,
position, k));

            % Extract Y-axis data
            y = normalized_data(:, (k-1)*3 + 2);
            x = 1:length(y);

            % Detect peaks for gait cycle
            [pks, locs] = findpeaks(y, 'MinPeakProminence', 0.5);
            [troughs, trough_locs] = findpeaks(-y, 'MinPeakProminence', 0.5);

            % Select troughs with Y value below -1.4
            valid_troughs_idx = find(-troughs < -1.4);
            selected_troughs = -troughs(valid_troughs_idx);
            selected_trough_locs = trough_locs(valid_troughs_idx);

            % Identify Swing and Stance phases
            swing_phases = [];
            stance_phases = [];

            for idx = 1:length(selected_trough_locs)
                prev_peak_idx = find(locs < selected_trough_locs(idx), 1, 'last');
                next_peak_idx = find(locs > selected_trough_locs(idx), 1, 'first');
                if ~isempty(prev_peak_idx) && ~isempty(next_peak_idx)
                    swing_phases        =        [swing_phases;        locs(prev_peak_idx),
selected_trough_locs(idx), locs(next_peak_idx)];
                end
            end
```

```
            for idx = 1:length(locs)-1
                is_part_of_swing = any(swing_phases(:,1) == locs(idx) | swing_phases(:,3) ==
locs(idx+1));
                if ~is_part_of_swing
                    stance_phases = [stance_phases; locs(idx), locs(idx+1)];
                end
            end

           % Signal sign inversion to invert reference system
            y = -y;

            % Graph with phase coloring
            hold on;
            colored_regions = false(size(y)); % Vector Tracking Colored Regions

            % Color the Swing phase
            swing_plot = plot(nan, nan, 'g', 'LineWidth', 2); % Placeholder for the legend
            for s = 1:size(swing_phases, 1)
                idx_range = swing_phases(s, 1):swing_phases(s, 3);
                plot(x(idx_range), y(idx_range), 'g', 'LineWidth', 2);
                colored_regions(idx_range) = true; % Mark segments as colored
                % Add vertical dotted lines at the beginning and end of the swing phase
                xline(x(swing_phases(s, 1)), 'k--', 'LineWidth', 1);
                xline(x(swing_phases(s, 3)), 'k--', 'LineWidth', 1);
            end

            % Color Stance Phase
            stance_plot = plot(nan, nan, 'r', 'LineWidth', 2); % Placeholder for the legend
            for s = 1:size(stance_phases, 1)
                idx_range = stance_phases(s, 1):stance_phases(s, 2);
                plot(x(idx_range), y(idx_range), 'r', 'LineWidth', 2);
                colored_regions(idx_range) = true; % Mark segments as colored
                % Add vertical dashed lines at the beginning and end of the stance phase
                xline(x(stance_phases(s, 1)), 'k--', 'LineWidth', 1);
                xline(x(stance_phases(s, 2)), 'k--', 'LineWidth', 1);
            end

            % Colorize the Y signal only in uncolored segments
            y_plot = plot(nan, nan, 'k', 'LineWidth', 1.5); % Placeholder for the legend
            % Original signal in black only where it does not belong to any phase
            uncolored_segments = find(~colored_regions); % Find uncolored segments
            segment_start = uncolored_segments([true; diff(uncolored_segments) > 1]); % Start
of segments
            segment_end = uncolored_segments([diff(uncolored_segments) > 1; true]); % End of
segments
            for seg = 1:length(segment_start)
                plot(x(segment_start(seg):segment_end(seg)),
y(segment_start(seg):segment_end(seg)), 'k', 'LineWidth', 1.5);
            end

            % Add the legend
            legend([swing_plot, stance_plot, y_plot], ...
                {'Swing Phase', 'Stance Phase', 'Y-Signal'}, 'Location', 'best');

            hold off;
            xlabel('Samples');
            ylabel('Acceleration [m/s^2]');
            grid on;

            % Select only Y-axis columns
            y_data = dati_filtrati(:, [2, 5, 8]); % Extract Y-axis data only

            % Feature Calculation
            % Duration of the Stance and Swing phase
            stance_durations = (stance_phases(:,2) - stance_phases(:,1)) / fs; % Duration in
seconds
            swing_durations = (swing_phases(:,3) - swing_phases(:,1)) / fs; % Duration in
seconds
```

```
            % Cadence: number of steps per minute
            num_steps = length(selected_trough_locs);
            duration_seconds = num_campioni / fs;
            cadence = (num_steps / duration_seconds) * 60; % Steps per minute

           % Inizializza array per memorizzare i campioni appartenenti alle fasi stance e swing
            valid_acc = [];

           % Estrai i campioni delle fasi di stance
            for s = 1:size(stance_phases, 1)
                valid_acc = [valid_acc; y(stance_phases(s, 1):stance_phases(s, 2))];
            end

            % Estrai i campioni delle fasi di swing
            for s = 1:size(swing_phases, 1)
                valid_acc = [valid_acc; y(swing_phases(s, 1):swing_phases(s, 3))];
            end

            % Mean and standard deviation of acceleration (on Y columns only)
            mean_acc = mean(y_data)/100; % Average over the three acquisitions
            std_acc = std(y_data)/100; % Calculate standard deviation for each column separately

            % Ratio of Swing to Stance Duration
            if ~isempty(stance_durations) && ~isempty(swing_durations)
                swing_stance_ratio = mean(swing_durations) / mean(stance_durations);
                swing_gait_ratio    =    mean(swing_durations)    /    (mean(swing_durations)+
mean(stance_durations));
            else
                swing_stance_ratio = NaN; % Avoid division by zero

            end

            % Feature Structure
            features = struct();
            features.mean_acc = mean(mean_acc); % Final average between acquisitions
            features.std_acc  =  mean(std_acc);  % Final  mean  of  standard  deviation  (for  all
columns)
            features.stance_duration = mean(stance_durations); % Medium stance duration
            features.swing_duration = mean(swing_durations); % Medium swing duration
            features.cadence = cadence; % Steps per minute
            features.swing_stance_ratio  =  swing_stance_ratio;  %  Ratio  of  Swing  to  Stance
Duration
            features.swing_gait_ratio = swing_gait_ratio;
            fprintf('\nRisultati per l''utente: %s all''acquisizione n. %u\n', utente, k);
            disp(features);


        end
    end
end
```

Figure 11: Code For Gait Cycle Phases Recognition With Accelerometer Placed On The Side Of The Leg

The code is intended to analyze the walking data acquired by an accelerometer on the side of the leg. The initial parameters are set the same as those relating to the previous analysis except for the *position parameter* which is set to '*side*'.
The code includes *carica_dati*, *fitered_signal,* and *standardized_signal* functions that are similar to those used for the walk cycle analysis with a sensor positioned on the front side of the leg.
The main code loops to iterate over all users. The name of the corresponding data file is constructed and the carica_dati function is invoked to extract the data from it. If data loading fails, a warning is displayed and the code moves on to the next iteration. The uploaded data is then preprocessed with low-pass filtering and Z-Score standardization.
Next, we analyze separately three acquisitions present in the file.

Unlike the case where the sensor is positioned frontally, the Y-axis is that of the direction of walking and therefore contains the most significant data for analysis. Similarly, the other axes do not contain significant information.

Therefore, for each acquisition, it extracts the Y-axis values and locates peaks (local maximums) and valleys (local minimums) with a minimum prominence of 0.5. Only valleys with a value of less than -1.4 are then selected to ensure that they are significant points in the gait cycle. Starting from these valleys, the phases of the step are determined: the "Swing" phase is identified as the segment between the peak before and after each selected valley, while the "Stance" phase is identified between two consecutive peaks, excluding those already involved in a "Swing" phase.

During the data processing phase, it was decided to invert the sign of the signal to optimize and to make the representation of the graphs more consistent, facilitating the interpretation of the results. This choice was motivated by the fact that the reference system used for data acquisition was inverted with respect to the standard convention.

A plot the data is created for each acquisition and shows the normalized Y-axis signal.

The Swing phases are colored green and the Stance phases red, with dotted lines to mark the beginning and end of each phase. Signal segments that do not belong to any phase are displayed in black.

The extracted features are similar to the case previously analyzed.

4 RESULTS

4.1 Gait Analysis – TOP

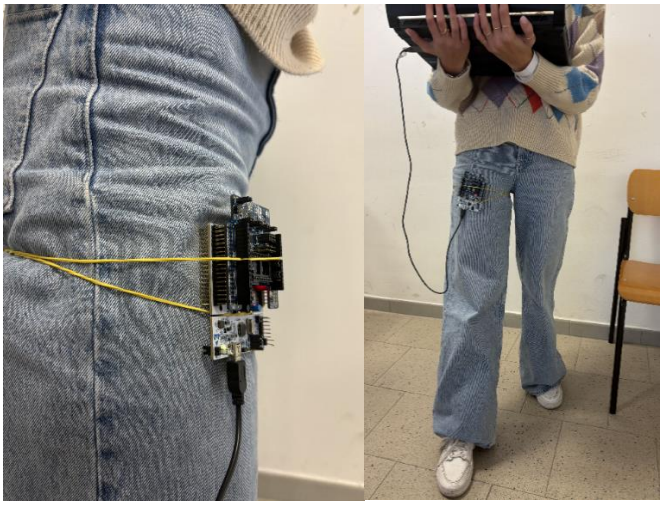

Figure 12: Wearable Sensor Positioned On The Top Of The Leg

For the gait analysis, the focus was on the collection and analysis of data relating to leg movement (Figure 10). To do this, we used a sensor, placed on the subject's leg using rubber bands, to acquire information in real time and ensure sufficient stability to acquire precise data. The card, connected to a computer, allowed the signal to be recorded while it was in motion.

The graphs shown in

Figure 13 represent normalized signals along the Z-axis derived from data acquisitions for different users. Each graph, relating to a different participant, highlights the trend of the signal over time by reporting the time in seconds on the x-axis and the normalized value of the signal on the y-axis.

In the various graphs, the lines in:

- black, represent the initial/final phase of the walk, or the sections of the gait cycle that are not correctly recognized;
- green, represent the swing phases of the gait cycle;
- red, represent the stance phases of the gait cycle;
- black vertical dotted lines, represent the breakdown of the phases for better visualization.

The analysis of the graphs obtained during the experiment made it possible to identify the different phases of the gait cycle. The identification of the stance and swing phases was carried out by identifying the peaks present within the signal by setting two thresholds: the minimum distance between the peaks equal to 20 samples and a minimum value equal to 1.5 in order to discriminate the peaks from small non-significant fluctuations in the signal. These values were chosen after a visual analysis of the plotted signal and trying some possible values.

The peaks correspond to the maximum acceleration, which typically occurs in the middle of the swing phase when the leg reaches maximum forward extension. The valleys, on the other hand, represent the minimums of the signal, which coincide with the moment when the foot touches the ground and the body begins to transfer its weight to the supporting leg.

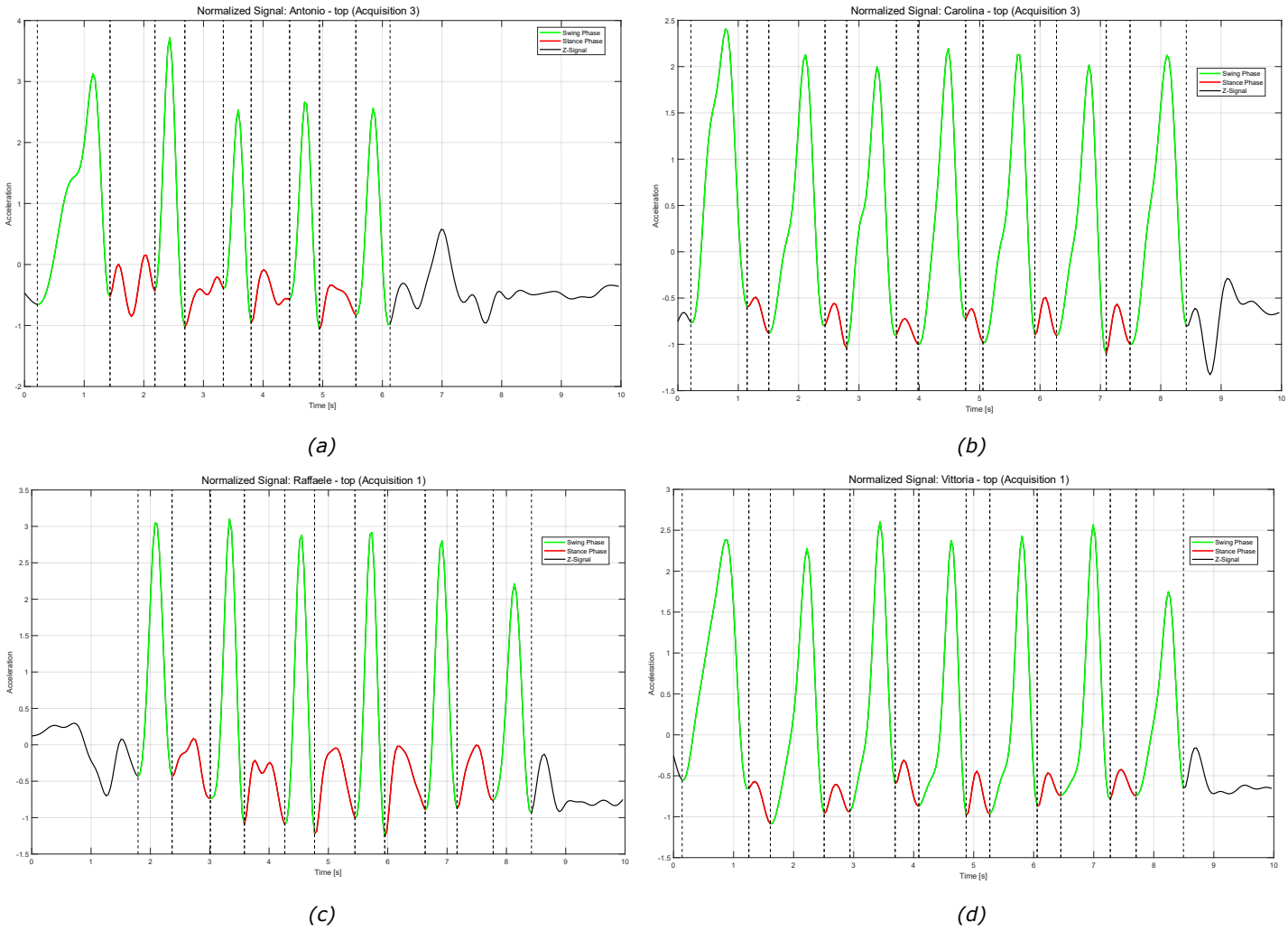

Figure 13: Gait Cycle Recognition For The Best Acquisition Of Different Users

From the charts in Figure 13(b) e (d), respectively Carolina's third acquisition and Vittoria's first acquisition, a fluid and continuous walk clearly emerges, with a regular alternation between stance and swing. On the contrary, in Figure 13(a), Antonio's acquisition, more marked variations in the amplitude of the peaks are observed, suggesting a less uniform gait. Plot (c), belonging to Raffaele, also shows a certain periodicity, but with more pronounced oscillations than (b) and (d), highlighting a greater variability in movement.

The amplitude of the signal further reflects these differences. In plots (b) and (d) the peaks appear constant and symmetrical, indicating a uniform thrust in the swing phase, while in (a) and (c) there are more evident fluctuations, suggesting a less regularity of the step.

The duration of the stance and swing phases varies among the different subjects analyzed. In graph (a), the stance is more extended, indicating greater foot contact with the ground before lifting, while in graphs (b) and (d) the transitions between the two phases are more rapid, making the movement more dynamic. Graph (c), on the other hand, shows a slightly more extended stance than (b) and (d), suggesting a slight slowdown or different weight distribution during walking.

These results are consistent with the physical characteristics of the subjects involved: graphs (b) and (d) refer to Carolina and Vittoria, who have a shorter foot, while graphs (a) and (c) belong to Antonio and Raffaele, who have a longer foot. This difference could influence stride

dynamics, with longer feet favoring longer ground contact and a more controlled walk, while shorter feet allow faster transitions between stance and swing.

In addition, considering that subjects with shorter feet also tend to have a shorter height and thus a shorter leg length, it is plausible that they take a greater number of steps in the same time interval to maintain a similar walking speed as subjects with longer legs. This aspect suggests that stride frequency might be inversely related to leg length, with shorter individuals adopting a higher cadence to compensate for reduced stride width.

In Table 1 The extracted traits and detailed metrics for each user and each acquisition are represented.

The analysis of the characteristics extracted from the data acquired by accelerometer required a careful evaluation of the signal quality. Some acquisitions have been removed from the Table 1 as they had negative average acceleration values. This phenomenon has been attributed to the method of attaching the accelerometer to the leg, which, having taken place by means of elastic bands, may have introduced micro-movements and slippages, altering the recorded data. Removed acquisitions will be considered later for a more in-depth evaluation.

For Antonio, only the third acquisition is available, in which an average acceleration of 0.6087 m/s^2 is observed, with a standard deviation of 2.4949 m/s^2, suggesting a moderate variability in movement. The duration of the Swing phase is 0.6523 s, while the Stance lasts 0.6631 s, resulting in a Swing/Stance ratio close to unity (0.9838). This value indicates a good balance between the two steps of the gait. The recorded cadence is 30 steps per minute, which is in the middle range compared to other users.

Vittoria shows a tendency towards a more dynamic and marked walk. Its cadence varies from 36 to 42 steps per minute, with average acceleration values always positive and between 0.7768 and 1.1065 m/s^2. The standard deviation of acceleration is among the highest in the group, reaching 2.8325 m/s^2 in the second acquisition, which suggests greater variability in leg movement. The Swing phase is consistently longer than the Stance, with a Swing/Stance ratio ranging between 2.1237 and 2.2386, indicating a clear prevalence of the swing phase over the stance.

Carolina stands out for a constant and balanced gait, with a uniform cadence of 42 steps per minute in acquisitions 1 and 3 and slightly lower (36 steps per minute) in the second. The average acceleration always remains positive, with values between 1.8086 and 1.9751 m/s^2, while the standard deviation is the smallest among all users (between 1.9388 and 2.0529 m/s^2), suggesting a smooth and fluid walk. Its Swing/Stance ratio is among the highest, fluctuating between 2.2386 and 2.8163, which indicates a significantly longer leg swing time than stance time.

Two acquisitions are available for Raffaele, characterized by a lower cadence than other users (24 and 30 steps per minute). The mean acceleration is positive but more contained than in the other subjects (0.1457 and 0.1016 m/s^2), and its standard deviation varies from 1.7241 to 2.3506 m/s^2, suggesting a moderate variability of movement. The Swing/Stance ratio increases from 1.2439 to 1.7057, indicating an increase in the relative duration of the Swing phase compared to the Stance. The duration of the Swing phase, in fact, increases up to 0.8100 s in the third acquisition, while the duration of the Stance remains around 0.4749-0.4898 s.

User	Acquisition	Mean Acc (m/s²)	Std Acc (m/s²)	Swing Dur (s)	Stance Dur (s)	Cadence (steps/min)	Swing/Stance Ratio	Swing/Gait Ratio
Antonio	3	0.6087	2.4949	0.6523	0.6631	30	0.9838	0.4959
Vittoria	1	0.7768	2.5136	0.8500	0.4002	42	2.1237	0.6799
	2	1.1065	2.8325	0.8244	0.3799	36	2.1698	0.6845
	3	1.0632	2.5387	0.8184	0.3656	36	2.2386	0.6912
Carolina	1	1.9751	2.0529	0.9421	0.3345	42	2.8163	0.7380
	2	1.8086	1.9388	0.8184	0.3656	36	2.2386	0.6912
	3	1.9592	1.9908	0.8705	0.3524	42	2.4697	0.7118
Raffaele	2	0.1457	2.3506	0.6093	0.4898	24	1.2439	0.5543
	3	0.1016	1.7241	0.8100	0.4749	30	1.7057	0.6304

Table 1: Gait Features Extraction - Top

4.2 Gait Analysis – SIDE

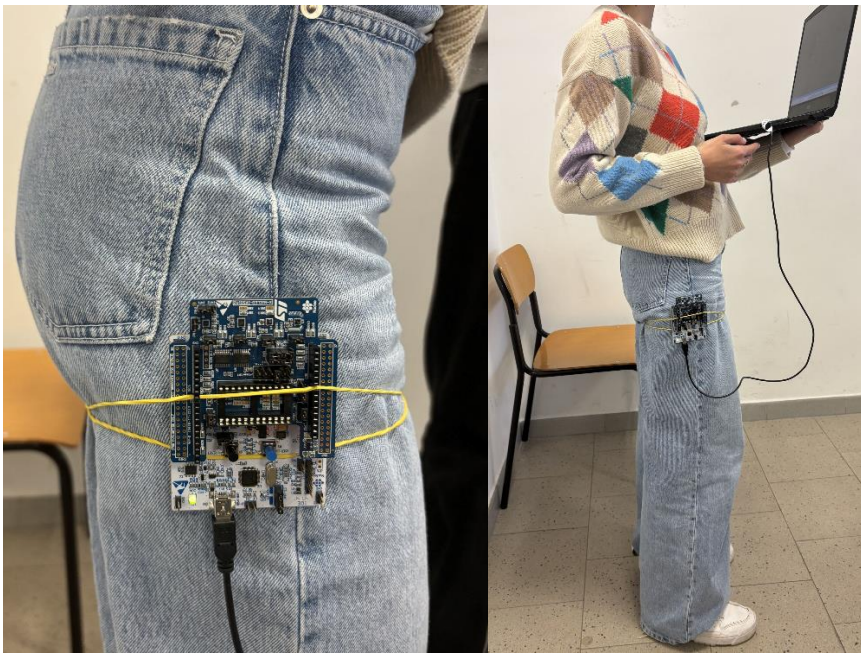

Figure 14: Wearable Sensor Positioned On The Side Of The Leg

For gait analysis, we proceed by focusing on the collection and analysis of data relating to walking by placing the sensor on the lateral part of the subject's leg (Figure 11).

Analysis of the graphs (Figure 15) obtained during the experiment made it possible to distinguish the different phases of the stride cycle. The identification of the stance and swing phases was done by detecting the peaks and valleys of the signal, applying a minimum prominence threshold of 0.5 to exclude less significant fluctuations. In addition, only valleys with values less than -1.4 were considered for more accurate selection.

Swing phases were determined by associating each selected valley with the immediately preceding and following peaks, while stance phases were identified by excluding intervals already assigned to swing. This method ensured a more accurate segmentation of the pitch cycle, making the analysis more reliable.

In this analysis, the peaks correspond to the maximum acceleration, which generally occurs in the middle of the swing phase, when the leg reaches maximum forward extension. Valleys, on the other hand, represent the minima of the signal, coinciding with the moment when the foot touches the ground and the body weight begins to transfer to the supporting leg. The same criterion was also adopted in the previous analysis, confirming the consistency in the interpretation of the different phases of the stride cycle.

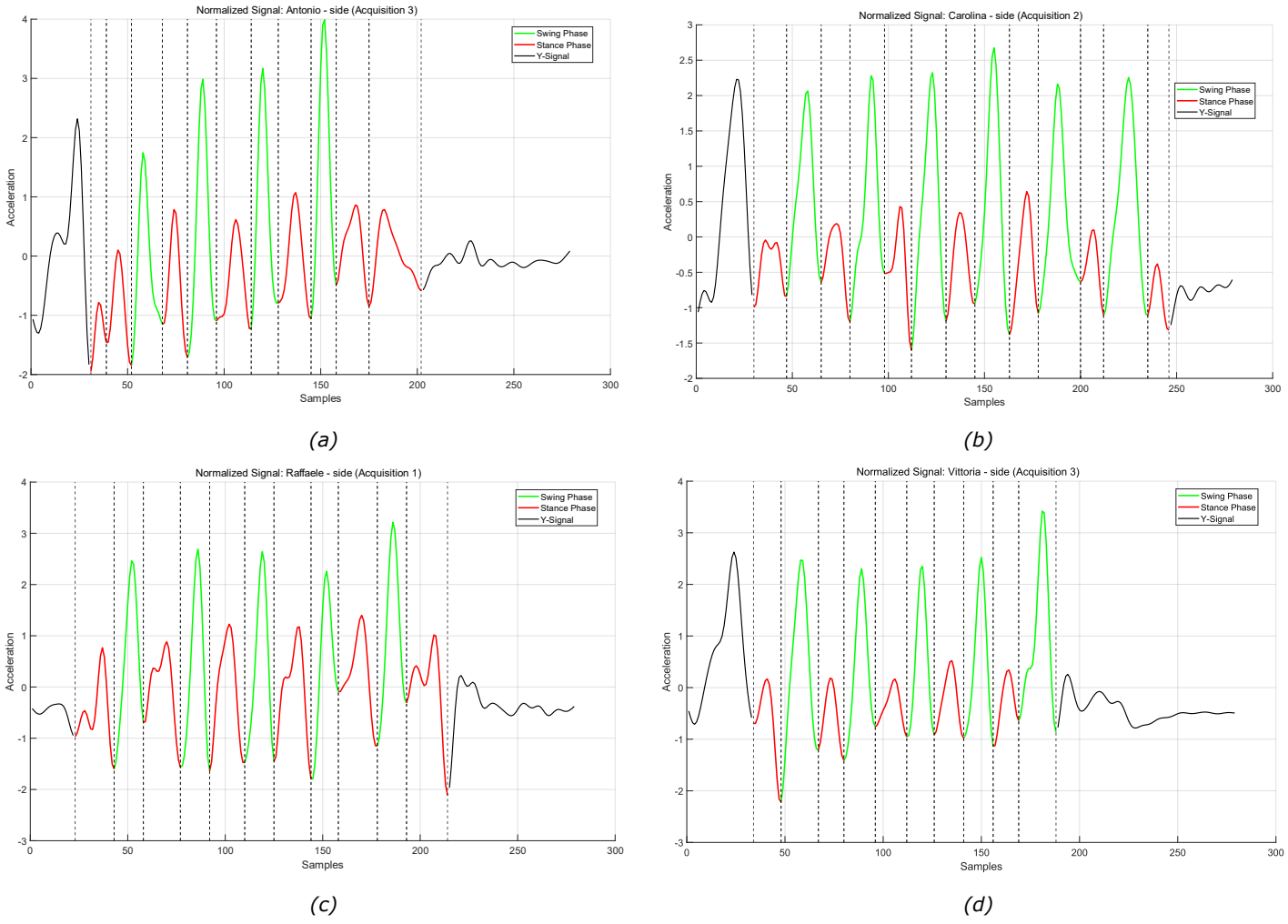

Figure 15: Gait Cycle Recognition For The Best Acquisition Of Different Users

Looking at the corresponding graphs, it is possible to make preliminary observations for each user.

In the case of Antonio (Figure 15 (a)), for the first acquisition, acceleration peaks are observed during the swing phase with rather marked amplitude variations, a sign of a less uniform oscillation of the leg and possible differences in the thrust or rhythm of the step. In addition, the stance phase appears more extensive in some cycles, indicating prolonged contact of the foot with the ground before lifting, which contributes to a less regular walk, with changes in speed or strength between steps.

On the contrary, Carolina's gait (Figure 15 (b)), is smoother and more continuous. The swing and stance phases alternate in a regular and harmonious way, with almost constant amplitude peaks, an indication of a stable support and a well-balanced thrust. The transition between

stance and swing appears quick and smooth, giving the walk a dynamic and constant character. A similar behavior can also be observed in the graph of Vittoria (Figure 15 (d)), where the regularity and symmetry of acceleration peaks suggest an extremely stable load distribution and pace timing, with fluid and well-defined transitions between the stance and swing phases.

The graph related to Raffaele (Figure 15 (c)), instead, shows an intermediate behavior. Although the gait has a certain periodicity, the oscillations of the signal appear more pronounced than in Carolina and Vittoria, suggesting greater variability in thrust and weight distribution. The duration of the stance phase is also slightly longer, a sign of more extensive contact with the ground, which may reflect a slightly more controlled and cadenced pace.

Table 2 shows the extracted traits and detailed metrics for each user and each acquisition.

The analysis of the characteristics extracted from the data recorded by triaxial accelerometer, positioned on the lateral part of the quadriceps, showed significant differences between users. Acquisitions with negative average acceleration values were excluded, as they were potentially affected by artifacts related to sensor fixing. The analyzed data therefore show the most representative acquisitions of gait dynamics.

Vittoria has a generally regular and constant pace, characterized by a stable cadence at 36 steps/min in all acquisitions. The average acceleration, although positive, is rather contained, suggesting a moderate propulsive thrust. The variability of movement is relatively low, indicating good walking control and a gait free of significant oscillations. This set of characteristics describes a stable and well-balanced walk.

Carolina shows a controlled and fluid gait, with an average acceleration close to zero indicating a balanced pace, without sudden acceleration or deceleration. The cadence is high, equal to 42 steps / min, a sign of a rapid but regular rhythm. The relatively long Stance phase suggests a greater focus on gait stability, helping to maintain good control while stalling. These characteristics outline a stable and well-managed walk, with a quick step but without obvious irregularities.

Raffaele shows a walk characterized by a more pronounced propulsive thrust than the other users, with higher average acceleration values. The Swing phase is prolonged, while the Stance phase is shorter, indicating a more dynamic walk, in which the forward thrust is favored over the stance time. Its cadence varies between 30 and 36 steps/min, suggesting some flexibility in the rhythm. Overall, Raffaele shows a more energetic and propulsion-oriented walking style.

Unlike other users, the acquisitions related to Antonio were excluded from the analysis due to the poor quality of the data recorded. The measurements were inconsistent, with negative and potentially distorted mean acceleration values. These anomalies were attributed to possible problems in accelerometer fixation, which may have introduced artifacts that compromised the reliability of the measurements.

User	Acquisition	Mean Acc (m/s^2)	Std Acc (m/s^2)	Swing Dur (s)	Stance Dur (s)	Cadence (steps/min)	Swing/Stance Ratio	Swing/Gait Ratio
Vittoria	1	0.1525	1.0241	0.5496	0.5806	36	1.0565	0.5137
	2	0.0695	1.0704	0.4480	0.6237	36	1.3920	0.5819
	3	0.0504	1.1614	0.5090	0.5950	36	1.1690	0.5390
Carolina	2	0.0146	1.2034	0.5069	0.6989	42	1.3788	0.5796
Raffaele	1	0.1190	1.1840	0.6989	0.5305	30	0.7590	0.4315
	2	0.1443	1.0241	0.6667	0.5161	36	0.7742	0.4364
	3	0.2272	1.0704	0.6380	0.5161	30	0.8090	0.4472

Table 2: Gait Features Extraction - Side

5 CONCLUDING NOTES AND POSSIBLE OBSERVATIONS

5.1 Observations

During the acquisition and analysis of the signals with the different sensors, some problems emerged that influenced the quality of the data and, therefore, the results obtained.

The acquisition card operated at a frequency of 12 Hz, however when the signal was displayed the sampling rate was 27.9 Hz because a sampling interval was not set during the acquisition phase. During the analysis phase, we considered downsampling to reduce the amount of data, but concluded that it was not necessary. The acquired signal already passes through a 3 Hz filter, which means that the high-frequency components have been attenuated and the information content is concentrated in the low frequencies. In addition, the number of samples recorded is not such as to make the processing too complex or computationally expensive. Reducing the frequency further would have resulted in a loss of potentially relevant information, with no real benefit to the analysis. For these reasons, we have chosen to keep all the acquired samples and analyze them in their original form, without applying any additional transformation.

Another critical issue occurred with the accelerometer sensor, which was fixed to the leg by a series of rubber bands. This method of attachment did not guarantee a perfect adhesion to the surface of the body, causing unwanted movement of the sensor relative to the leg. As a result, some acquisitions were less accurate, with acceleration readings affected by small sensor slippages.

The present acquisition in Figure 16 it is in fact an example of a low-quality acquisition due to the poor fixing of the accelerometer by means of rubber bands. The peaks in the Stance phase have a similar amplitude to the peaks in the Swing phase, making it ineffective to recognize the two phases and extract features.

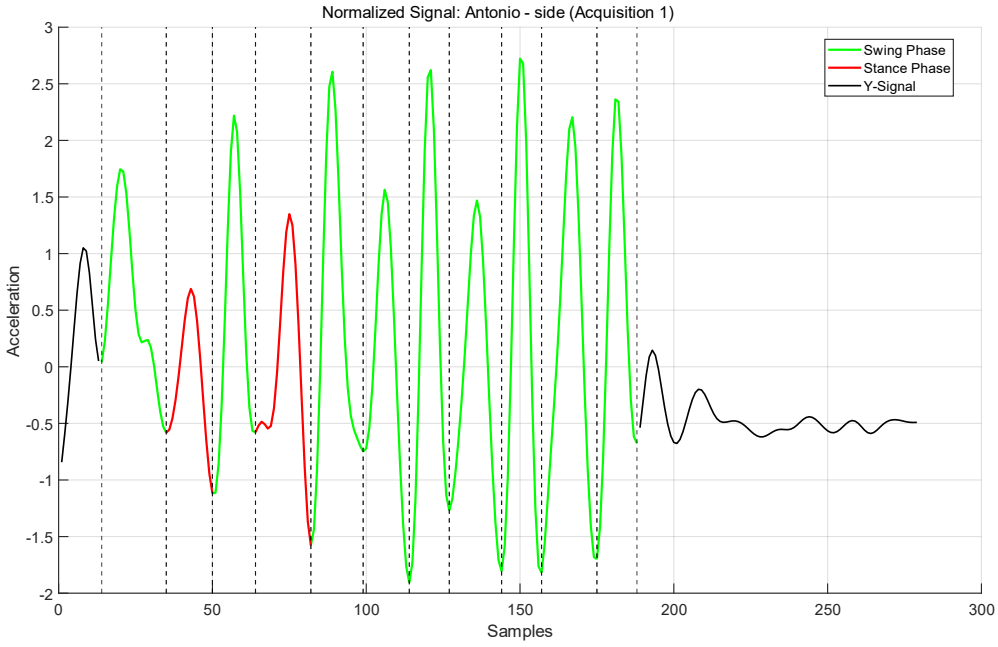

Figure 16: Incorrect Gait Recognition Analysis Due To Poor Quality Acquisition

Si osserva che i dati raccolti dal sensore posizionato sulla parte laterale della gamba in Table 4 e sul lato frontale Table 3, mostrano un segnale disturbato, con un evidente influsso del contributo gravitazionale che si traduce in una segmentazione delle fasi di camminata meno chiara e affidabile. La classificazione delle fasi di swing e stance risulta meno coerente, il ritmo della camminata non viene rappresentato in maniera corretta e, di conseguenza, il valore delle feature estratte non è ottimale.

User	Acquisition	Mean Acc (m/s^2)	Std Acc (m/s^2)	Swing Dur (s)	Stance Dur (s)	Cadence (steps/min)	Swing/Stance Ratio	Swing/Gait Ratio
Antonio	1	-0.5092	2.5022	0.5161	0.6631	30	0.7784	0.4377
	2	-0.5758	2.5882	0.6093	0.4898	24	1.2439	0.5543
Raffaele	1	-0.5600	2.5025	0.5556	0.6595	36	0.8424	0.4572

Table 3: Wrong Features Extraction - Top

User	Acquisition	Mean Acc (m/s^2)	Std Acc (m/s^2)	Swing Dur (s)	Stance Dur (s)	Cadence (steps/min)	Swing/Stance Ratio	Swing/Gait Ratio
Antonio	1	0.1372	1.2034	0.5914	0.5615	54	0.9495	0.4870
	2	0.0381	1.1587	0.3719	0.4500	54	1.2102	0.5475
	3	-0.0383	1.1840	0.5786	0.5197	30	0.8982	0.4732
Carolina	1	-0.1559	0.9849	0.5735	0.6870	42	1.1979	0.5450
	3	0.0594	1.1587	0.6165	0.7348	36	1.1919	0.5438

Table 4: Wrong Features Extraction - Side

5.2 Conclusion

The laboratory experience has allowed us to deepen the process of acquisition, processing and extraction of features from physiological signals, in particular gait. 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 gait recognition, highlighting the main challenges related to their implementation and data acquisition.

Gait analysis for user recognition is an innovative tool to uniquely identify people through the peculiarities of their journey. In this context, the analysis of motor patterns, such as the transitions between the swing and stance phases and the acceleration parameters, allows to capture individual characteristics, making it possible to accurately differentiate the subjects. To further increase the reliability of this technique, it is desirable to refine the segmentation of signals and adopt machine learning algorithms capable of learning and adapting to individual variations dynamically. In addition, the integration of additional sensors and the development of real-time analysis systems represent promising directions to improve the accuracy of gait recognition.

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.