Improving Human-Robot Interaction In Wearable Robotics Through Comprehensive User Interface Design

BY

GREGORY SACCO
B.S., Politecnico di Torino, Torino, Italy, 2021

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2023

Chicago, Illinois

Defence Committee:

Myunghee Kim, Chair and Advisor
Miloš Žefran, Department of Electrical and Computer Engineering
Diego Regruto Tomalino, Politecnico di Torino
Stefano Quer, Politecnico di Torino

ACKNOWLEDGMENTS

First and foremost, I extend my thanks to my entire family for their unwavering support through every means possible, enabling me to achieve this accomplishment. I am grateful to both academic institutions, the professors, and the researchers of the RRLab who helped and provided me with the opportunity to work on this project. I extend my appreciation to all the individuals with whom I collaborated and/or created new friendships, both in Italy and in the United States.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF CODES

LIST OF ABBREVIATIONS

WBB Wii Balance Board

HIL Human In the Loop

AFO Ankle Foot Orthosis

HRV Hear Rate Variability

RMSSD Root Mean Square of successive Differences between normal heartbeats

SFB Soft Flexible Biopatch

CoP Center of Pressure

SUMMARY

Wearable robotics represents a transformative advancement in the field of assistive technologies, promising enhanced mobility and improved quality of life for individuals with mobility impairments. However, despite their potential, these emerging technologies are often challenging to use and inaccessible to a wide range of users, particularly medical professionals and patients. This thesis addresses the pressing need for user-friendly interfaces in wearable robotics, focusing on the development of two interfaces designed to enhance accessibility to complex systems. The first interface developed is for managing a controller for an active ankle exoskeleton, a sophisticated wearable device designed to assist with mobility. The challenge lies in creating a user interface that simplifies the control of this exoskeleton. The second interface supports a machine learning algorithm that personalizes the exoskeleton's control parameters, minimizing the patient's energy expenditure during specific physical activities while wearing the exoskeleton. The interface allows to control and visualize the outputs of this algorithm. Additionally, this thesis introduces a third software, implemented in Matlab, which connects to, calibrates, and collects data from a Nintendo Wii Balance Board. This affordable and reasonably accurate device has gained popularity for its ability to detect the Center of Pressure (CoP), indicating the patient's weight distribution during various activities. The culmination of this research represents a meaningful step forward in the field of wearable robotics. Through the successful integration of all three software components,this study demonstrates the effectiveness in using ad hoc

software to make these wearable robotics systems accessible to everyone. By addressing the challenges faced by users and simplifying the complexities of these technologies, this thesis contributes to the ongoing efforts in creating more accessible and user-friendly wearable robotics systems.

Chapter 1

INTRODUCTION

Wearable robotics, a cutting-edge field at the intersection of robotics and human augmentation, is gaining prominence as a transformative technology with the potential to revolutionize various domains. This emerging field focuses on the development of robotic devices that can be worn on the body to augment or enhance human capabilities. These wearable robotic systems aim to seamlessly integrate with the user, providing assistance, support, or even restoring lost functionalities. In recent years, the diffusion of wearable robotics has been accelerating, driven by advancements in materials, sensors, and control systems. The technology's applications span a wide spectrum, ranging from healthcare and rehabilitation to industrial and military sectors. Wearable robotic devices are designed to assist individuals with mobility impairments, enhance physical performance, and optimize ergonomics in professional settings.

1.1 Motivations

The motivation for undertaking this research comes from the engagement in a thesis project within the Rehabilitation Robotics Laboratory (RRLab) at UIC, whose leader is Dr. Myunghee Kim. The focus of this laboratory's research initiative is to leverage wearable robotics to enhance the rehabilitative capabilities of individuals experiencing partial mobility loss. The wearable devices developed by RRLab, coupled with efficient customization software, have demonstrated the ability to reduce users' effort during physical activities.

However, the complexity of these tools poses a significant hurdle, especially when considering their potential application beyond laboratory environments for experimental purposes. The end users of these technologies, namely individuals affected by disabilities or doctors utilizing the tools to aid patient recovery, underscore the critical need to make these instruments more accessible. The motivation for this thesis arises from this imperative.

The thesis comprises three key components: first is a MATLAB software capable of interfacing with a Nintendo Wii Balance Board, second is an application programmed using a commercial software for intuitive controlling an ankle exoskeleton, and third is an application facilitating user monitoring of the wearable robot optimization process. The latter serves as an interface designed to enhance the accessibility of the Human-in-the-Loop toolkit, a tool developed by P. Kantharaju et al. in [1]. The authors highlight that their optimization framework is primarily tailored for proficient

engineers, presenting a potential barrier for less-experienced users or non-researchers. Recognizing this limitation, the thesis aims to contribute to a more inclusive and user-friendly approach by designing intuitive interfaces for wearable robotics applications, making them accessible to a broader audience.

1.2 Thesis Structure

This thesis is organized into eight chapters. The second chapter provides contextual information about the systems developed preceding this work. Understanding these concepts is essential for comprehending the developments detailed in this thesis. Subsequently, we deepen the Implementation chapters, spanning from Chapter three to Chapter six. Within these sections, I elucidate the work and design choices defining the development of the three main topics of this thesis: the MATLAB software for Wii Balance Boards, the Control System Interface, and the Optimization Interface. The final chapter in this Implementation section focuses on elucidating how the new interfaces have been seamlessly integrated with the systems outlined in chapter two. Chapter 7 narrates the experiment conducted to validate all the systems developed in this work and their integration. Lastly, the conclusion synthesizes how these interfaces have effectively facilitated the experiment and suggests potential modifications and enhancements for the future.

Chapter 2

RELATED WORK

In order to develop the Optimization UI, several other systems have been used for testing and debugging. This chapter wants to describe these systems. Comprehending these tools is critical to understand the user interface and its functionality. This section will explain in details the design and purpose of these systems. The section is divided in Human In the Loop (HIL) toolkit, the Ankle Foot Orthosis (AFO) (both realized by the Rehabilitation Robotics laboratory of University of Illinois at Chicago[1]) and a biopatch electronic sensor for measuring high-quality ECG (designed by the Georgia Institute of Technology[2]).

2.1 Human In the Loop toolkit

The first component under discussion is the HIL toolkit, developed and detailed by P. Kantharaju et al. in [1]. This study successfully achieved the objective of creating an open-source, device-independent personalization framework that enhances the ac-

[1]https://rehab-robotics.lab.uic.edu
[2]https://www.gatech.edu

cessibility of HIL optimization. To maximize the effectiveness of wearable robots and enhance user comfort, customization of these devices is essential. However, commonly, optimization algorithms are tailor-made for the specific devices they are intended for, posing challenges in terms of accessibility. This study addresses this limitation by developing an algorithm capable of personalizing any device based on any physiological feedback. Moreover, the HIL toolkit addresses another significant issue. Many conventional optimization algorithms are often slow, a constraint particularly impactful when considering that the technology should be designed to support the elderly, injured, or individuals with lower physical capabilities. Real-time optimization is crucial, and the physical effort required from the user must not be prolonged. Kim et al. implemented Bayesian optimization, a noise-tolerant, sample-efficient, and non-parametric method, achieving notable improvements. They successfully obtained two parameters for controlling an exoskeleton during a squat activity within 15 minutes.

These customization algorithms rely on physiological feedback. Han et al. [2] and Jeong et al. [3] demonstrated that muscular activity can be employed as input for HIL optimization. Various other signals and feedback have been implemented and proven efficient for HIL optimization [4] [5]. In this paper, two types of feedback were predominantly utilized: indirect calorimetry feedback, which measures the inhaled oxygen and exhaled carbon dioxide of the subject and ECG signal from a heart sensor, the functionality of which is explored in a subsequent chapter. As for actuators, three different devices were implemented: ankle-foot prosthesis, robotic ankle exoskeleton, and

personalization of gait parameters (step frequency). Through the utilization of diverse feedback and devices, the researchers demonstrated the versatility of this framework. The HIL toolkit comprises three components, as illustrated in Figure 1.

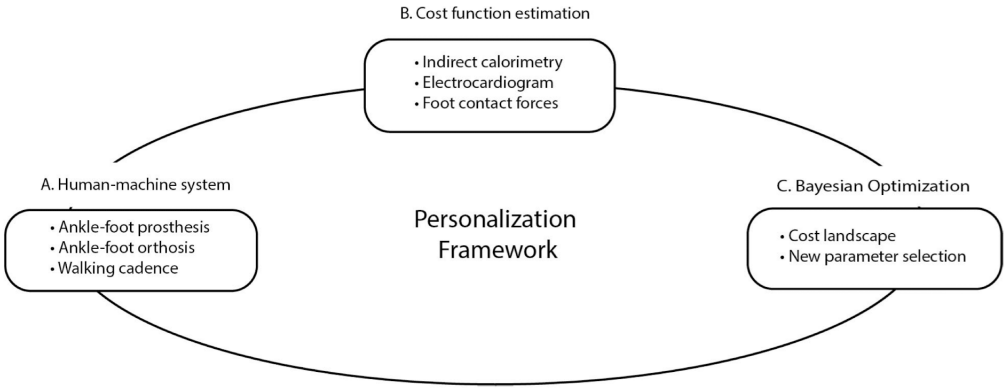

Figure 1: HIL framework's overview. Screenshot from [1]

The initial module is denominated the Human-Machine System, encompassing the wearable robots or commands from a device optimized to reduce the physical exertion of the patient. Examples elucidated in this paper encompass an ankle-foot prosthesis, a robotic ankle-foot orthosis, and an acoustic device capable of emitting sounds to command the patient's step frequency. The second component is the Cost Function Estimation, a software segment designed to interpret physiological feedback signals derived from dedicated sensors. It is imperative in this segment to explicate the features of the signal concomitant with physical effort. These features facilitate the assessment of energy expended by the patient during activity and, crucially, enable the quantification of the assistance effect provided by the wearable robot. Examples of feedback for cost estimation are indirect calorimetry, electrocardiogram, and foot contact forces.

The third constituent of this framework is Bayesian Optimization based on two pro-

cesses. The initial process, called cost landscape, employs the Gaussian process to predict the posterior distribution based on data and the use of kernel functions. The kernel function is responsible for calculating the mean and standard deviation of the posterior distribution from real cost function samples. Various types of kernel functions exist; however, for this framework, the squared exponential kernel has been employed. The subsequent process is New Parameter Selection that is possible thanks to the Acquisition function which is derived from the mean and standard deviation of the posterior function. This function guides the search in selecting points in the objective function domain during the optimization process. Its purpose is to balance exploration (the exploration of new regions in the domain) and exploitation (the exploration of regions near where the function appears to be optimal). In essence, it is responsible for choosing the next point to measure in the real cost function approximation. Different types of Acquisition functions exist; in this framework, the Expected Improvement (EI) was utilized. The repetition of these two processes (depicted in Figure 2) facilitates the approximation of the real function. This framework underwent testing in three distinct scenarios, incorporating varied types of feedback and devices. Across all instances, a noteworthy reduction in physical exertion was achieved when comparing the Optimal condition to the No device condition or General condition (the user has assistance from non-personalized device).

while Not converged **do**
Measure cost (y_i) for parameter ($\mathbf{x}_i$)
$y_i^* \leftarrow y_i + \epsilon,\ \epsilon \sim \mathcal{N}(o, \sigma_{noise})$ ▷ Gaussian Noise
$\mathbb{D} \leftarrow y_i^*, x_i$
Posterior function
$\mu(\mathbf{x}_*) = k_n \cdot (K + \sigma^2_{noise} \cdot \mathbb{I})^{-1} \cdot \mathbf{y}_{1|n}$ ▷ Mean of Gaussian Regression
$\sigma(\mathbf{x}_*)^2 = K - k_n \cdot (K + \sigma^2_{noise} \cdot \mathbb{I})^{-1} \cdot k_n^T$ ▷ standard deviation of Gaussian Regression
Acquisition function
$V_{\mathbf{x}_*} \leftarrow acq(\mu_{x_*}, \sigma_{x_*})$ ▷ Generate the value for acquisition function
$x_{n+1} \leftarrow argmax(V_{\mathbf{x}})$ ▷ Selecting the parameter with the large value as next parameter
Convergence criteria
if time > experiment time $\parallel x_{n+1} == x_n == x_{n-1}$ **then**
Converged
else
Not Converged
end if
end while

Figure 2: Pseudo-code for computing posterior distribution. x_* are all the points of the sample space. k is the kernel function. k_n is a vector get by applying k on known data points x_n. K is a matrix obtained by applying k on the entire sample space. σ_{noise} is the hyper-parameter tuned during the optimization. Screenshot from [1]

2.2 Ankle Foot Exoskeleton

A fundamental device for the research of this thesis is the AFO. This is the mechatronic device that will give assistance to the patient in doing the squatting activity. The device is thoroughly described in [6] and it is a valuable solution to improve research in wearable assistive robotics whose goal is to improve the gait of individuals with reduced mobility. The paper cited above stresses the importance of developing a device that can also mechanically adapt to the variations in body measurements and walking mechanics, while taking into account the operational aspects of the device determined by the user's physical state and objectives in gait training.

The mechanical design is made of two main parts: the Exo-foot and Exo-Tibia. Both

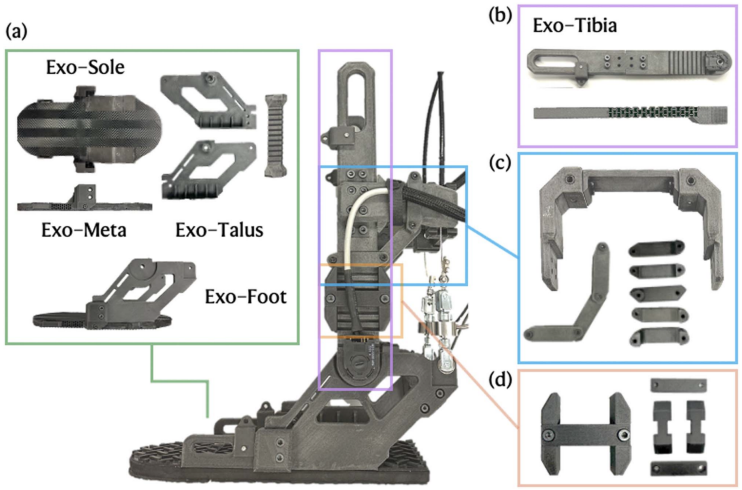

Figure 3: AFO's modular components. Screenshot from [6]

of these components can easily be replaced with other similar parts whose dimensions, however, meet the characteristics of the patient. Exo-Foot is the bottom part composed by the sole and additional components in order to connect the Exo-Foot with the Exo-Tibia. The latter is important to attach the exoskeleton to the patient's tibia and make sure that they are securely fixed. On the Exo-Tibia there is connected an additional mechanical supporting piece in order to hold and route the cables coming directly from the motors. The additive manufacturing technique was used to develop the parts mentioned above. This allowed them to make lighter and more anthropometrically efficient components. The material used is nylon-carbon fiber N12 Carbon Fiber and the AFO is around $0.9kg$ in weight.

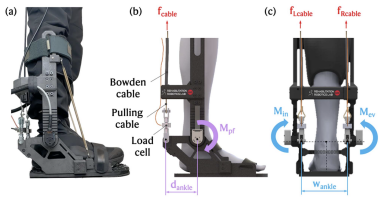

Figure 4: Subject wearing the AFO on the right leg. Figure from [6]

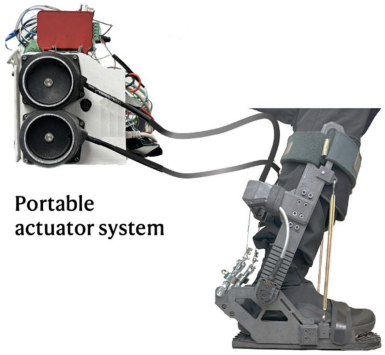

Figure 5: Portable actuator system in the off-board emulator setting with the AFO worn by a subject. Figure from [6]

The portable actuator system which controls the system explained above has two optical encoders (HEDS-5500-A06, Broadcom Inc., CA, USA). They are located in the joints between Exo-Foot and Exo-Tibia. One in the medial side and the other in the lateral side. They are able to measure the angles of the joints in the sagittal plane. Torque feedback is given instead by two DYMH-103 tensile load cells. The torque is given by two motors (EC i-52s Maxon Group, Switzerland). Two EPOS4 50/15 EtherCAT motor drivers (Maxon Group, Switzerland), on the other hand, take care of controlling the motors and reading the feedback signals coming from the encoders and load cells. The output torque of the AFO is controlled through a Simulink real-time controller.

The motors are engaged in supporting plantar flexion. While for dorsiflexion, the device uses two elastic bands connected between the Exo-Tibia and Exo-Foot. Two small and quick-response limit switches are placed on the medial and lateral sides of the AFO in order to do not exceed the maximum expected plantarflexion angle for security pur-

poses. As shown in the Figure 5, the motors can transmit torque to the exoskeleton through two $2m$ long Bowden cables. The diameter of the cables is $5mm$. The weight of the whole system shown in the Figure 5 is about $6.5kg$.

A portable dual cable-driven system was designed in order to offer assistance with plantarflexion and in/eversion. With the following equations we can define the torques applied for plantarflexion and in/eversion assistance:

$$M_{pf} = M_L + M_R \tag{2.1}$$

$$M_{in-ev} = (M_L - M_R) \cdot \frac{w_{ankle}}{2d_{ankle}} \tag{2.2}$$

where M_L and M_R stands for torque caused by the motor connected to the left and right side of the exoskeleton. D_{ankle} and W_{ankle} are dimensions taken from the geometry of the Exo. It is possible to check these two features in Figure 4.

The controller is run in Simulink (Mathworks, MA, USA) at the frequency of $1KHz$. The controller is divided into mid level control and low level control. The mid level part describes the angle/torque characteristic. So it is responsible for defining the plantar and in/eversion torque needed to assist the human subject. The low-level controller, on the other hand, is responsible for translating the desired torques into velocity commands to be sent to the motors. The latter controller is of the proportional type and its output,

that is the command in velocity, is defined by the Equation 2.3.

$$\dot{\theta}_d = k_{gain}(M_d - M_m) \tag{2.3}$$

where M_d and M_m are desired torque and torque measured by the load cells. The Mid level controller can work in different modes: *No control* to completely disable the motors and controller, *Position control* to enable the controller and set the desired torques to zero. This enables to cancel the loosening of the transmission cables. Finally, we have the *Squat control* mode, which instead is used for squatting activity and thus to define the torque needed to support the patient during the various phases of the squats. In Figure 6 it is possible to observe an approximate schematic of the characteristic that represents the torque/angle characteristic used by the controller in this mode.

The characteristic is described by 6 states but here for simplicity we define three

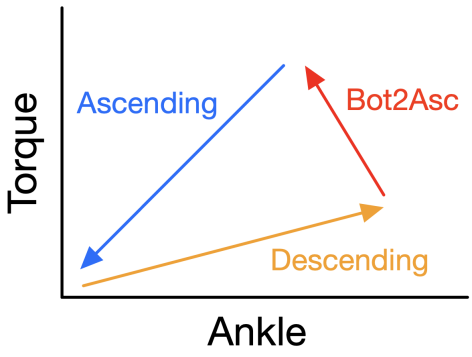

Figure 6: Torque-angle characteristic

main steps: *Descending*, *Bot2Asc* and *Ascending*. For *Descending* and *Ascending* we have a function that determines the desired torque based on the angle. The difference

between these two steps is that they can have different stiffness, and therefore, different resulting slopes in the characteristic. The red arrow instead represents the *Bot2Asc* state that stands for "Bottom To Ascend" state. This state is critical to transition as smoothly as possible from descending to ascending. An immediate transition from descending to ascending, assuming that $K_{ascending}$ is greater than $K_{descending}$, could cause exaggerated resistance at the conclusion of the descending movement.

In the article, the authors also tested the whole system and obtained satisfactory results. The tests in a laboratory setup indicated that the device could generate $40Nm$ of torque for plantarflexion and $16Nm$ of torque for inversion/eversion movements, achieving rise times of $70ms$ for plantarflexion, $77ms$ for eversion, and $84ms$ for inversion. The control bandwidth for torque exceeded $13Hz$ for both plantarflexion and inversion/eversion. During a squat assistance task involving a human participant, the device was able to closely reach the desired torque pattern, displaying a maximum average root mean square (RMS) error of $2.9 \pm 1.2Nm$ for plantarflexion assistance and $0.7 \pm 0.5Nm$ for inversion/eversion assistance.

2.3 ECG biopatch sensor

As observed in the preceding chapters, within this field, the utilization of a HIL framework is of paramount importance in order to ensure the personalizing process of a wearable robot capable of assisting the subject. A fundamental constituent of the HIL framework is the Biofeedback signal, which furnishes us with insights into the

vital functions of the human subject. Specifically, biofeedback aids in comprehending the level of physical effort experienced by the patient during a given activity. One of the most commonly employed methodologies for this purpose entails estimating the metabolic cost via indirect calorimetry and a respiratory mask, which measures the inhaled oxygen and exhaled carbon dioxide of the subject. Nonetheless, despite its reliability, the aforementioned system is also notably inconvenient, bulky, and impractical. Indeed, devices of this nature necessitate a rigid mask that fits snugly onto the face without gaps and are equipped with a small backpack (worn by the subject) housing the battery system and antenna for data transmission. These components can affect the subject's mobility. These cumbersome systems directly contradict the goal of developing wearable robots aimed at assisting individuals in performing repetitive motions. These systems entail a protracted setup, are confined to laboratory settings, and offer discomfort due to their substantial dimensions and weight. Notably, this system mandates a minimum of 3 minutes to estimate the required physiological signal.

An alternative solution, on the other hand, involves employing sensors capable of interpreting the Electrocardiogram (ECG). The ECG signal is not perfectly periodic [7]; it varies according to the effort experienced by the patient [8], [9], [10]. Greater effort corresponds to a more periodic heart rate. The variable that describes this characteristic is known as Hear Rate Variability (HRV). To compute and quantify this variable, Root Mean Square of successive Differences between normal heartbeats (RMSSD) is used. RMSSD involves taking the square root of the sum of squared intervals between

adjacent normal heartbeats. The time intervals are measured between adjacent peaks of the ECG and are referred to as RR intervals, as depicted in Figure 7.

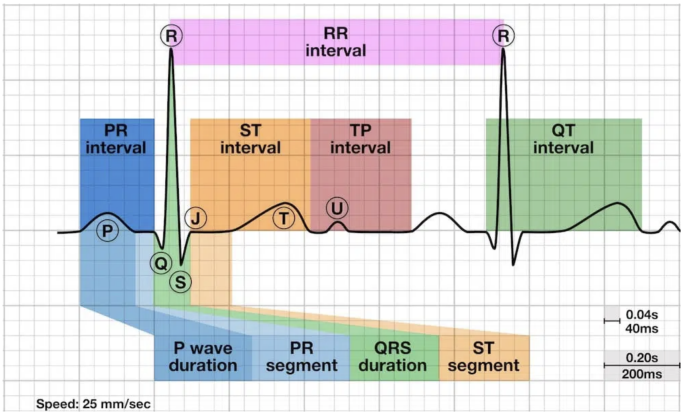

Figure 7: ECG signal description. Credits https://litfl.com/t-wave-ecg-library/

Numerous commercial devices are available in the market capable of measuring HRV; however, these systems often entail discomfort during wear, thereby imposing limitations on mobility. In [11] the authors delineate a study centered on the development of a soft, flexible biopatch proficient in collecting ECG data and transmitting them via Bluetooth. The core objective of this article resides in disseminating the design of the Soft Flexible Biopatch (SFB) and elucidating the outcomes of several experiments that underscore the robust negative correlation between normalized metabolic cost and normalized HRV-RMSSD.

The SFB consists of multiple layers. The initial layer, which makes contact with the skin, comprises electrodes. Subsequently, an adhesive base is present and on top of that there is a flexible Printed Circuit Board (fPCB) and the power supply. The fPCB incorporates all the necessary electronics for data processing. Specifically, within its

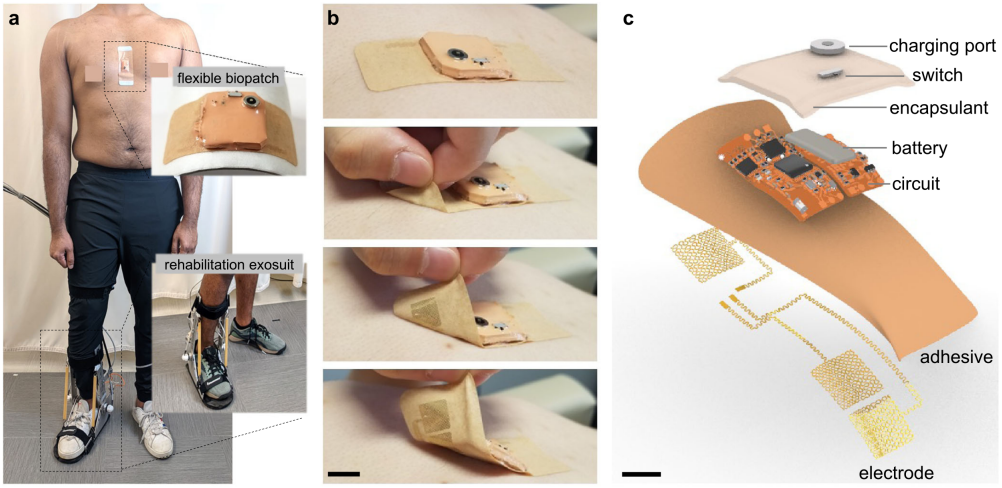

Figure 8: Soft flexible bioelectronic system (SFB) integrated with an ankle-foot-orthosis (AFO) exoskeleton. Figure from [11]

structure, components such as the ECG analog-to-digital converter (ADS1292, Texas Instruments), Microprocessor (NRF 52832, Nordic), and a high-frequency low-power Bluetooth antenna are encompassed. As depicted in Figure 8, the fPCB is encapsulated by a soft elastomer (Ecoflex™30, Smooth-On), enabling enhanced mechanical and electrical resistance. Atop of this encapsulation, the charging port and a switch are positioned. Notably, the device is equipped with a $3.7V$, $40mAh$ battery that ensures 9 hours of uninterrupted operations. A complete recharge is accomplished within 30 minutes. Due to skin irregularities, the device must exhibit flexibility of more than 15° to maintain consistent contact with the sternum. This property has been validated through comprehensive mechanical testing, yielding no damage with regards to mechanics, electronics, or overall performances.

The assessment of SFB quality hinges upon obtaining a reliable ECG signal. To this end, the sensor records raw data which are subsequently filtered using a first-order

Butterworth band-pass filter. Cut-off frequencies are $0.5Hz$ and $60Hz$. The Signal-to-Noise Ratio (SNR) is measured at 25 dB, facilitating clear distinction of ECG signal characteristics, in particular the R peaks, across all usage scenarios: squatting, walking, and running. Subsequent to filtering, the signal undergoes convolution through a moving average process, followed by application of an RMS envelope to eliminate noisier periods. They developed an HR detection algorithm that compares the values of the peaks with the threshold shown in Equation 2.4.

$$Threshold = NoiseLevel + 0.25(SignalLevel - NoiseLevel) \tag{2.4}$$

If the value is higher than the threshold, the peak is identified as a valid R peak. This threshold depends on two parameters: NoiseLevel and SignalLevel. They are dynamically updated after every classification with Equation 2.5 and Equation 2.6. If the valid R peak is higher than the result of Equation 2.5, SignalLevel will be updated with Equation 2.5. If not, NoiseLevel will be updated with Equation 2.6.

$$SignalLevel = 0.125Peak + 0.875SignalLevel \tag{2.5}$$
$$NoiseLevel = 0.125Peak + 0.875SignalLevel \tag{2.6}$$

After this post processing of the ECG data, a python library called Neurokit2 [12] has been used in order to calculate the HRV-RMSSD values.

In summary, this study presents a compact wearable bioelectronic system integrating a flexible biopatch and an ankle-foot exoskeleton. The compact biopatch replaces bulky tools, offering accurate metabolic rate measurement. The conformal device en-

sures close skin contact for precise ECG and HRV-RMSSD recording. Unlike traditional mask-based calorimetry, this wearable setup, comprising SFB and AFO, exhibits strong performance in various activities with a high signal-to-noise ratio ($>$25 dB) and robust Pearson R correlation (-0.758, p-value: $1.2e-7$) with steady-state metabolic cost.

Chapter 3

IMPLEMENTATION - Wii Balance Board

The Wii Balance Board (WBB) is a peripheral device developed by Nintendo for the Wii console. It was released in 2007 as part of the Wii Fit game package. The Balance Board is a rectangular platform equipped with multiple sensors that can detect shifts in weight and balance. The primary purpose of the Wii Balance Board is to enhance game play and promote physical activity. It allows players to engage in various interactive exercises and games that focus on fitness, balance, and coordination. The board can measure a player's center of gravity and movements in real-time, providing feedback and tracking progress. As already discussed in chapter 2, WBBs are a very cheap alternative for Biomedical applications. In terms of data precision, these systems might not attain the level of accuracy demonstrated by force plates, yet they frequently achieve satisfactory results. Our goal was to be able to detect the pattern drawn by the Center of Pressure (CoP) of each foot of the subject during squatting activity. This information can be used in two possible ways. The first is to use the CoP information in real time as a BioFeedBack signal. This real time data could be the input for an

optimization algorithm like the Bayesian Optimization. In particular, the idea is to compute a cost function using the CoP data and by minimizing this cost function, we can optimize the selection of the control parameters for every subject. The second is to simply compare the CoP data coming from different conditions of squatting during the experiment. For example we can compare the CoP data when the subject is squatting in a *no device* condition and the CoP data when the subject is in a *optimal* condition. For *optimal* condition we mean that the Human subject is squatting wearing the ankle exoskeleton with personalized assistance. By comparing, we can see if the assistance given by the exoskeleton can positively affect the kinematics of the squatting movement. Throughout this section, we will go over the implementation of the software able to connect, calibrate and retrieve the data coming from the WBBs.

3.1 Wii Balance Board - hardware

Figure 9: Wii Balance Board from Nintendo. Credits: https://it.wikipedia.org /wiki/File:Wii_Balance_Board_transparent.png

The working principle is very simple. The device is a platform of dimension 23x43cm

and it has 4 load cells in the 4 corners. The sensors are cantilevered metal bars with a strain gauge that converts the applied force into a voltage.

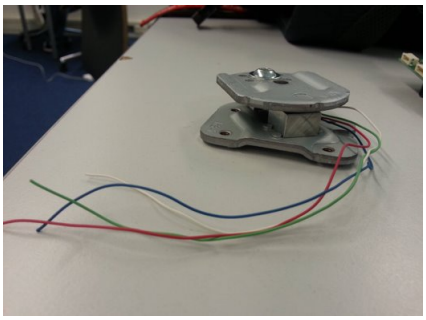

Figure 10: Extracted pressure sensor. Credits: https://electronics.stackexchange.com /questions/83861/connect-wii-balanceboard-pressure-sensor-to-an-arduino

Each bar has two strain gauges as represented in Figure 12. When a force is applied to this system, the bar bends and we get a deformation of the two strain gauges. One will tend to become narrower and the other will become wider. These deformations are translated into a voltage signal for the electronic unit of the WBB. The cantilevered bars are made of duralumin, an alloy with high strength-to-weight ratio. So the straining on the metal is very slight. For example, if a weight of $100kg$ is applied, the bar would only bend by $0.1mm$. However, it is precise enough that it can accurately detect weight differences of 500 grams. As per the available literature, the Wii Balance Board is reported to sample each force channel at a notably high frequency of around $100Hz$. This sampling rate exceeds the recommended minimum of $50Hz$ for accurately capturing the Center of Pressure (COP) during postural sway assessments [13]. The literature also suggests that the force sensors utilized in the WBB are designed to

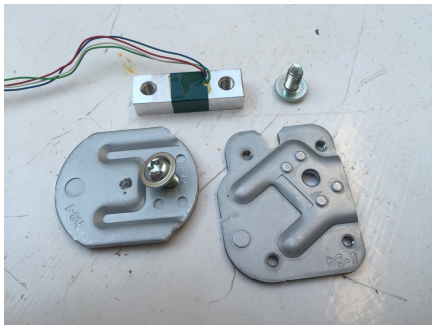

Figure 11: Disassembled pressure sensor. Credits: https://www.xsimulator.net/community/threads/diy-load-cell-brake-pedal-short-tuto.6042/page-2

be linear and demonstrate COP noise levels of approximately $0.5mm$ [14]. It is also important to notice that because of how this balance is designed, it is possible to detect only vertical reaction forces [15]. This stands as one of the numerous reasons why force plates command a higher cost compared to Wii balance boards.

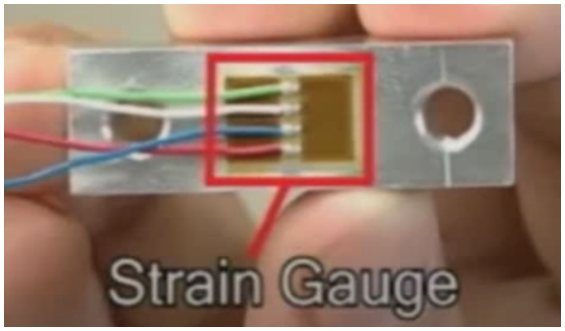

Figure 12: Strain gauge used in WBB's sensors. Credits [Video on http://www.rehabtools.org/wii-balance-board.html]

3.2 WiiBalance Library - software

Over the past few years, Nintendo devices have proven to be very inexpensive and fairly accurate solutions to be implemented in various research applications such as

biomedical or virtual reality. For this very reason, some research laboratories have been working on making these devices easier to use. In particular, the University of Notre Dame explained how they developed low-level programming to facilitate the connection between the Wiimote and Matlab [16]. WiiLab, as described in the paper, is a combined collection of C# and Matlab libraries for Windows that produce an intuitive API that greatly abstracts the difficulty of using the Wiimote. The Wii Remote, commonly referred to as the Wiimote, is the gaming controller for Nintendo's Wii system. In order to connect to the WBB instead, Pete R Jones, from University of London, developed another Matlab library called WiiBalance[1]. This library essentially incorporates additional functionalities to the WiiLab library, enabling it to establish connections not only with Wiimote devices but also with the WBB. For using these two libraries, it is important to have installed Matlab 2012 onward and 32 bit Windows as operating systems. On our computer with Windows 10 (64 bit), we used Matlab 2015 (32 bit) in order to run the library. Once the installation was completed, we performed a test using the library. This was the outcome of the example given by the WiiLab tool (Figure 14):

[1]https://github.com/petejonze/wiibalance

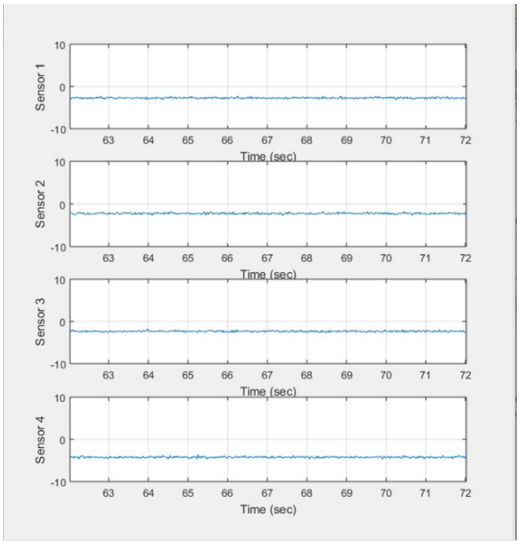

Figure 13: Extracted pressure sensor

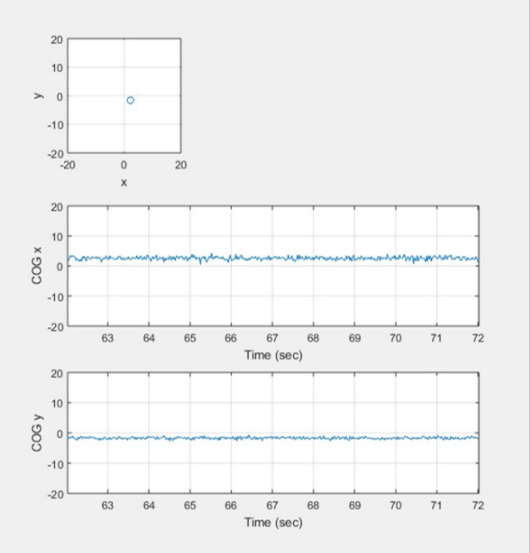

Figure 14: Disassembled pressure sensor

In Figure 14 there are 3 plots. The two on the bottom represents the coordinates (x and y) of the Center of Gravity (CoG) with respect to time. The vertical axes have centimeters as units of measurement. The first plot at the top instead represents the position of the CoG, indeed the x coordinate is on the horizontal axis (cm) and y coordinate is on the vertical (cm). Figure 13 shows 4 plots that we easily implemented in order to display the values read by each sensor in time. Because of the uncalibrated device and little knowledge of how the data is retrieved from the hardware, we couldn't understand the unit of measure of the sensors' outputs. However, the first important thing was to check the linearity of measurements from each sensor. Without linear sensors it would be very hard to compute the CoP. So we flipped the WBB with the sensors facing up and we applied several weights to each sensor and saved the values given. The Matlab script was developed to connect to the balance board and store single measurements from the sensors. The list of the weights applied was: 4.75, 9.5, 14.12 and

18.77kg. We applied the force using the system shown in Figure 15. The method for applying the weights is shown in Figure 16. In particular we used a rod with a ring in the middle. The ring is for blocking the weight disks that will be loaded on the rod. On one end of the rod there is a circular pointing tip made of steel whose diameter is 7mm.

Figure 15: Weight application device and weights

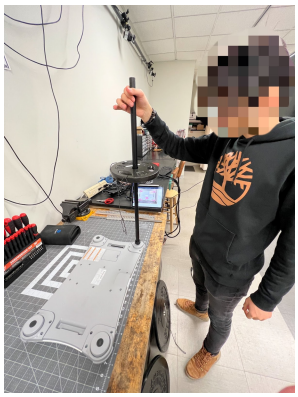

Figure 16: Load application method

The procedure was the following one. One disk at the time was applied using the custom pointing device shown in Figure 15. Once the disk was stable on the sensor we took the measurement. We did this on each sensor by measuring all the weights listed before. When the procedure was done, the scripts showed the plots with the given outputs. The result plots were all very similar like in Figure 17. On the horizontal axis we have the values of the weights applied to the sensors in kilograms, on the y axis instead, there are the respective values measured by the balance board. The red asterisks are the samples, while the blue line is a function interpolating them. The outcome is deemed satisfactory due to the fact that, despite the ambiguity of units of

measurement, the sensors exhibit a consistent linearity.

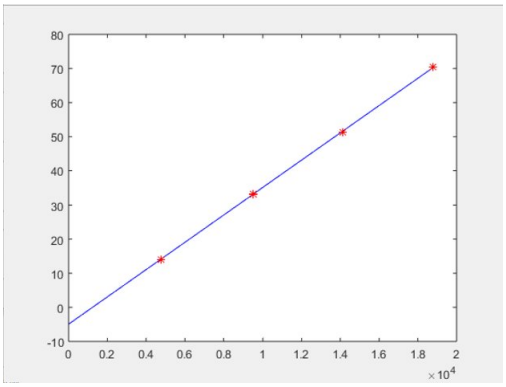

Figure 17: Linearity check on WBB's sensors

In light of the current circumstances, it has become increasingly important to recognize the significance of developing scripts specifically designed to calibrate the WBB. By doing so, we can effectively address and mitigate the challenges presented by this situation. Before working on the calibration functions we found out a problem in the computation of the CoP. The CoP, called CoG in the original library WiiBalance, was retrieved by a function that we couldn't open to get more information about:

```
1 CoG = obj.bb.wm.GetBalanceBoardCoGState();
```

Code 1: WiiLab function for computing CoP

However, we wanted to test it in order to understand its behavior. So with our custom application device we applied a force in a specific point on the balance. As expected, we got a coordinate measurement that was within the ranges related to the size of the Balance Board. However, by keeping the same position and sufficiently

increasing the pressure, we were able to move the point measured even outside the ranges. This is not tolerable for our application. We are interested in detecting the position where most of the force of the foot is applied. It is intuitively that this CoP point must be located inside the area where the foot is positioned. In the example shown before instead, by changing the force applied to the same point, we were getting different coordinates. The original goal of the WBB from Nintendo was to detect the balance, not the CoP. In order to change this we deleted that function. We used two formulas as defined in [14], [15]:

$$COP_x = \frac{L}{2}\frac{(TR + BR) - (TL + BL)}{TR + BL + TL + BL} \tag{3.1}$$

$$COP_y = \frac{W}{2}\frac{(TL + TR) - (BL + BR)}{TR + BL + TL + BL} \tag{3.2}$$

For these equations we considered the center of the balance as the origin and we used the its dimensions for L and W. In particular L is 43,3 cm and W is 22,8 cm. TL, TR, BL and BR refer to the sensors. They stand for top-right, top-left, button-left and button-right. Figure 18 represents the corners names and postions. Following the aforementioned change, the experiment was conducted again, similar to the previous test. Notably, even by increasing the applied weights, the center of pressure (CoP) remained unchanged if the point of force application remained constant.

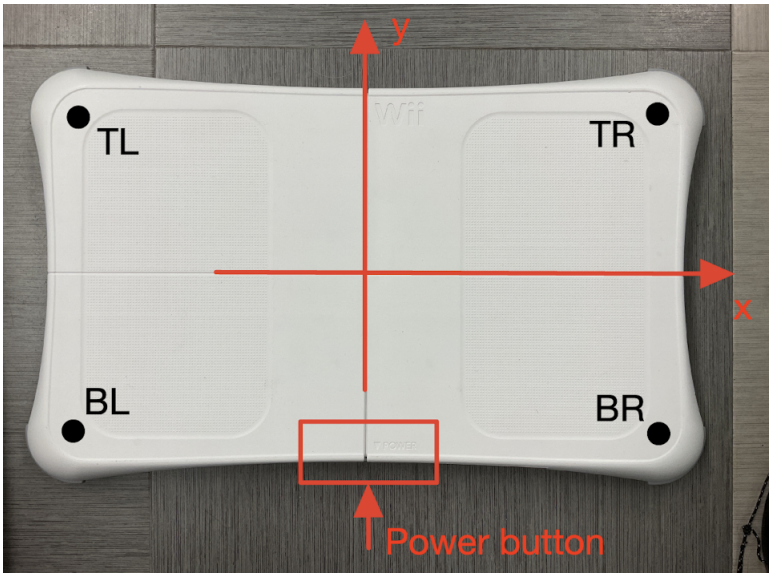

Figure 18: Sensors' labels

3.3 Calibration functions

From the experiment explained above it is clear that all the sensors were all approximately linear, however, the characteristics computed for each sensor were quite different in terms of slope and offset. To contribute to the most precise measurements possible, it is crucial that every sensor has exactly the same characteristic. In order to calibrate the balance board we designed three calibration functions. WiiBalance is a class that calls in its initialization function the WiiLab class. In order to add all the calibration functions a higher class called *WiiBwCalibration* has been created. This last class calls the WiiBalance library in its initialization function and it contains the three calibration methods.

The first function *NewtonTransform* has the goal to change the characteristic of each sensor to an ideal one. The ideal equation that a sensor needs to have is $y = 9.81 * x$.

This equation represents the formula to compute the gravitational force given a mass. The sensor characteristic inputs will be in kilograms and the outputs will be in Newtons. For example, by applying one kilogram on the sensor, we should read $9.81N$. The second calibration function is called *CalibrationWdrift*. As we said before, in the first calibration the WBB is flipped with the sensor facing up to make it easier to place the weights on the sensors. After this calibration, the sensors read zero when no force is applied. However, when we flip again the balance (WBB feet facing down), the sensors will perceive the weight of the balance itself. *CalibrationWdrift* is designed in order to measure the weight of the balance and subtract it from the next measurements. The third calibration is important for correcting the CoP equation.

3.3.1 NewtonTransform

The most complicated part for this calibration is the set up. It is important to maintain the weight application device still on the sensors and to make sure the order of the weights applied is respected. For this purpose, we designed a set up in order to respect these requirements. As shown in the Figure 19, we used a metallic parallelepiped shaped mechanical structure (available in the lab) and another metallic component. By combining these two mechanical supports we created a structure able to make the custom weight application device standing and still. Because of the height of this structure, we used a rigid black box on which the WBB is then placed along with a scale. In particular we placed the scale on the box and then we arranged the flipped WBB (sensors facing up) on the scale.

Figure 19: Setup for NewtonTransform calibration

The reason for using the scale is quite simple. It is crucial for this calibration function to know exactly the value of the weight we are applying, the absolute value. In fact, even if we know exactly the mass applied to the sensor, this value depends also on how the load is placed on the device. If it is not perfectly vertical, this value could be different. In this case, the weight force will break down into two components, vertical and horizontal components. The WBB, because of the way it is designed, cannot sense horizontal forces. This means we lose information. To resolve this problem, we decided to place a scale under the WBB. So every time we have to start the calibration on a sensor, we place the WBB on the scale, we make sure everything is stable, then push the Tare button on the scale. After this process we can start the calibration. The Matlab function asks the user to apply the first weight on the sensor, insert the value of the weight read by the scale in the command window and then press Enter to measure.

Next step is adding the second weight and measuring again. Continue to the last

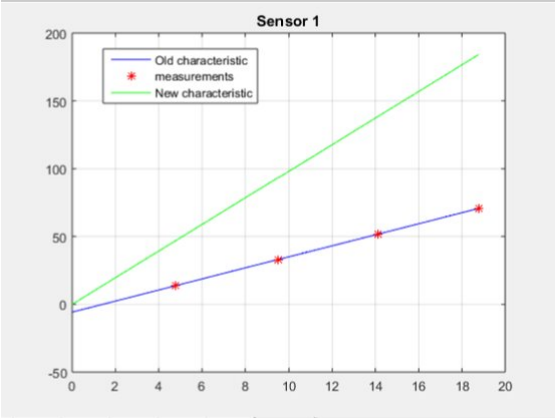

Figure 20: NewtonTransform result

weight that is 18.77 kg. After the last measure we get the plot showing the result (Figure 20). The red asterisks are the values measured and the blue line is the characteristic of the sensor before the calibration.

From the blue characteristic we can understand slope and offset. These two features will be used in order to get the green characteristic that shows the relationship between weights applied and values measured in Newtons after the calibration. The formula used to calibrate the sensor is:

$$Y_{new} = (Y_{old} - offset) \cdot \frac{9.81}{slope_{Y_{old}}} \tag{3.3}$$

Where Y_{old} is the blue line and Y_{new} is the green one. This means that after this calibration is performed, every single sensor measurement is subtracted from the offset value of its initial characteristic, and then the result is multiplied by a constant given by the gravitational acceleration over the slope of the characteristic prior to calibration. This step ensures that each new sensor measurement gives in output a value in newtons

that is perfectly consistent with the weight applied to the measuring device.

3.3.2 CalibrationWdrift

This calibration is performed immediately after the NewtonTransform function to delete the offset given by the weight of the balance. The user flips and places the WBB on the ground. It is important to mention that this calibration needs to be performed in the place where we want to conduct the experiment and use the calibrated WBB. After the execution of CalibrationWdrift, the balance board shouldn't be moved. This is because this calibration may be different based on the surface where we place the balance. When the device is in the desired position we can press Enter. The Matlab script will start recording the data coming from all the sensors. When little weight is applied to the balance we get very noisy signals from the sensors. In order to get a more accurate estimate of the data, the script records the signals for ten seconds. Once the data is recorded, it convolutes the signals with a one dimension digital filter. In particular we take the data as a one dimension vector and then we perform a convolution between this vector and the filter, that is a vector with one hundred components and every element is equal to 1/100. The design of the filter can be represented by the following code:

```
8 windowSize = 100;
9 b = (1/windowSize)*ones (1, windowSize);
```

Code 2: 1-D digital filter

The design consisted in tuning the value for *windowSize*. This value determines

the size of the filter and the magnitude of each element. Every component of this 1D digital filter is equal. Once the convolution is done the Matlab script prints the result in Figure 21.

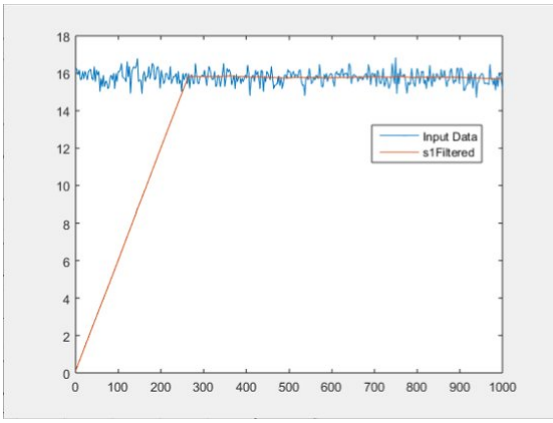

Figure 21: Raw data and filtered data from Sensor 1

The example in Figure 21 is for sensor 1. The sensor was reading a mean value of $16N$ because of the weight of the WBB (blue signal). For a larger value of windowSize we get a smoother signal in output but the filtered signal will take more time to converge to the real signal. So the design of this filter consisted in trying different values for windowSize in order to obtain a trade-off between smoothing of the signal and fast convergence. After several attempts, we decided the parameter to be equal to 100. We wanted to get a smooth output, but a fast convergence of the filtered signal was more important. If the delay for convergence is too long, it means that more samples will be neglected and that we have to collect more data, so the recording time should be longer. We wanted to keep the recording time to less than 10 seconds in order to make the calibration procedure faster.

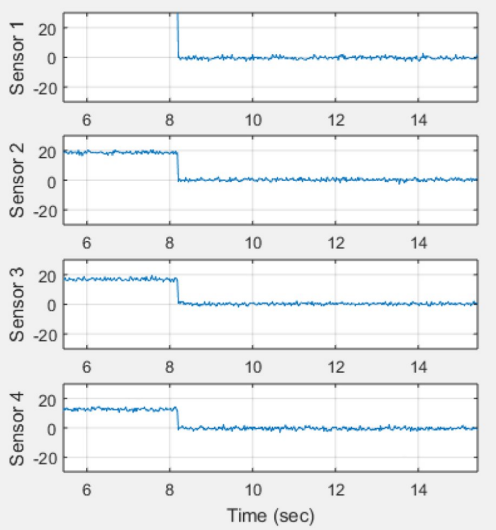

Figure 22: Second calibration result

The *FilterData* function, that is the function that performs this filtering, neglects the first two thirds of the filtered data and then takes the mean of the other samples. This procedure is performed on all the transducers. In Figure 22, we can see the effect of this calibration on the post calibration measurements. Because of the pressure given by the mass of the balance, all the sensors were giving in output values greater than zero. We filtered these noisy signals, took the mean and then we subtracted these correction values to the next measurements. As figured in the plots, all the sensors experience a force of $0N$ after the *calibrationWdrift* execution.

3.3.3 CalibrationP

For this method the position of the device remains as set after CalibrationWdrift. We exploited again the mechanical structure mentioned in the first calibration. This time we can not use the box, otherwise we would change the position of the balance.

We used the mechanical structure as it is shown in the Figure 23. The weight is always

Figure 23: CalibrationP setup

the same ($18.77kg$) and by moving the metallic support we were able to apply steadily the load in all the points without moving the balance. As can be seen in Figure 23, we covered the balance with transparent tape then we drew a grid on the tape with unit measurements in centimeters. With this grid we were able to mark the points where we wanted to measure. CalibrationP is for the formula that computes the position of CoP. The idea is to place our custom weight application device with test load in three specific points of the WBB. In each point we keep the weight application system stable and in the meanwhile the balance records the data. Once the data is recorded, the matlab function filters the signals like it has been done in the previous calibration function (same filter too) and it computes the errors between actual position and measured position for both x and y coordinates. The matlab script stores these two errors for all the positions and then computes the average error in x and y coordinates.

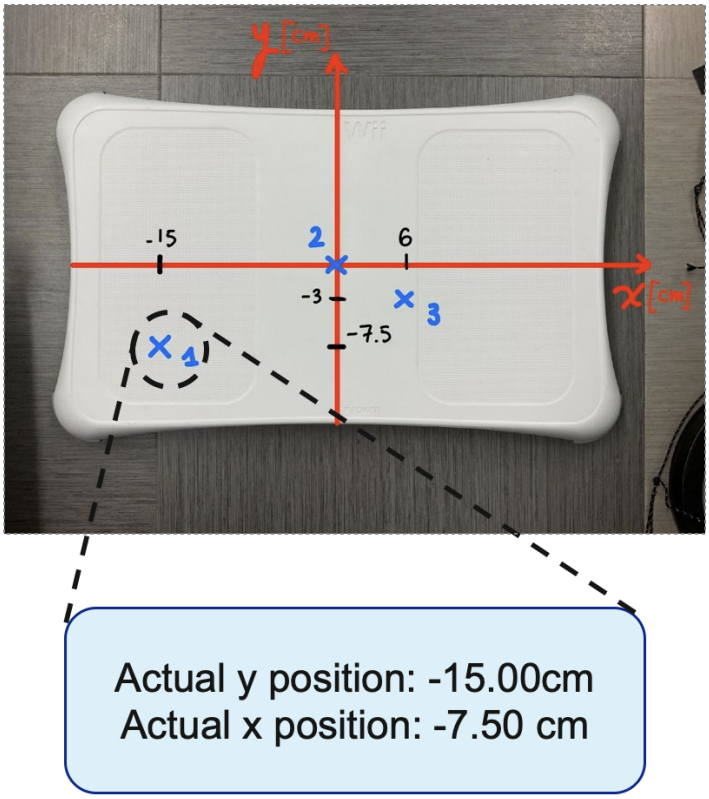

Figure 24: Map of points selected for calibration

These two averages will be the last correcting factors and they will be subtracted to the future calculations of the CoP. In Figure 24 it is possible to notice the positions we decided to explore for this calibration. With respect to the reference system, the positions [x,y] are: [-15, -7.5], [0,0] and [-3,6] where the unit of measure is centimeter and these coordinates have been chosen randomly. The points selected are represented by the blue crosses drawn on the balance. For all the positions we applied the weight and we collected the data. In order to evaluate the measurements we created histograms that plots the frequency of each value measured. Examples for position 1 are depicted in Figure 25 and 26.

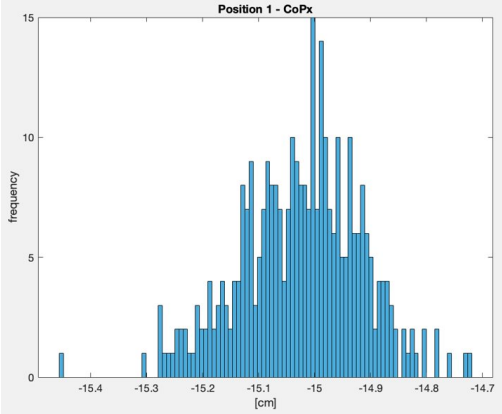

Figure 25: Measures of CoPx

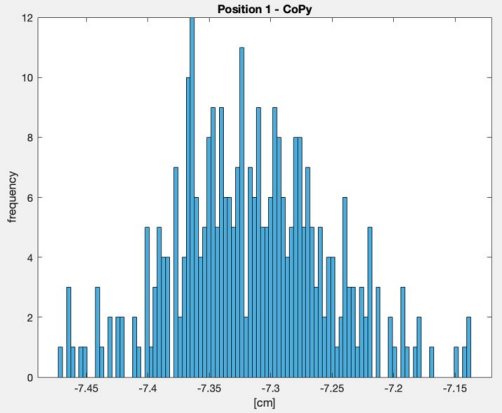

Figure 26: Measures of CoPy

In the Figure 23, we have the x coordinate measured by the balance. With mean equal to -15.0253cm and standard deviation of 0.109cm. For the y coordinate instead we got a mean value of -7.3155cm and standard deviation of 0.062 cm. We can therefore easily deduce errors of -0.1845cm for the vertical (y) coordinate and 0.0253cm for the horizontal (x) coordinate. For this example we can also show the signal analysis of the data coming from the sensors. Here we have the frequency of the Newtons measured by the transducers.

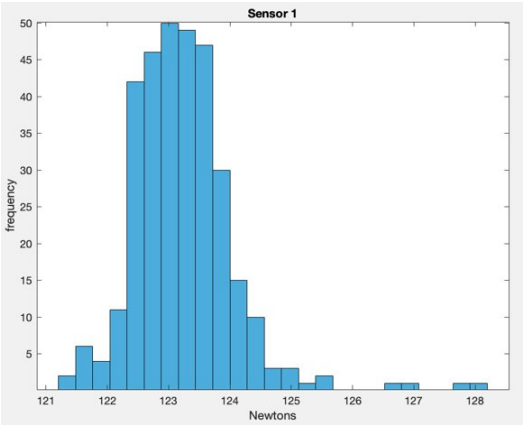

Figure 27: Measures from sensor 1

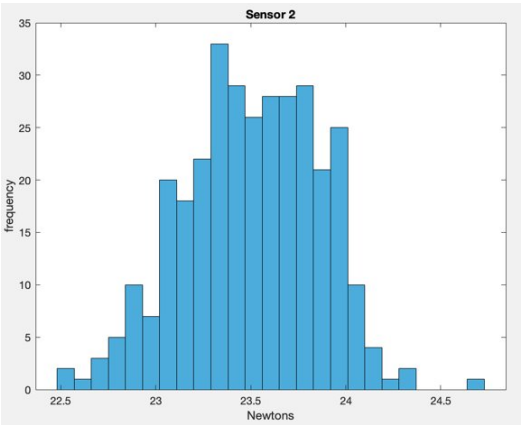

Figure 28: Measures from sensor 2

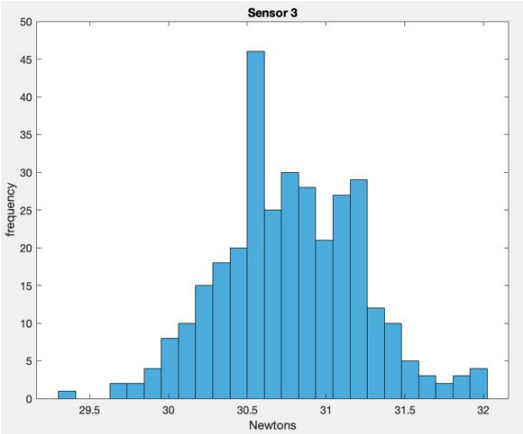

Figure 29: Measures from sensor 3

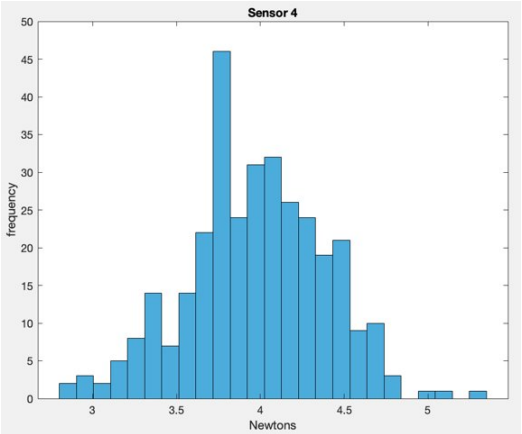

Figure 30: Measures from sensor 4

Sensor one is the closest to the point of application of the weight and it has the highest mean ($123.2447N$). Sensor four is the farest and it has the lowest mean ($3.9712N$).

3.4 Two Wii Balance Boards integration

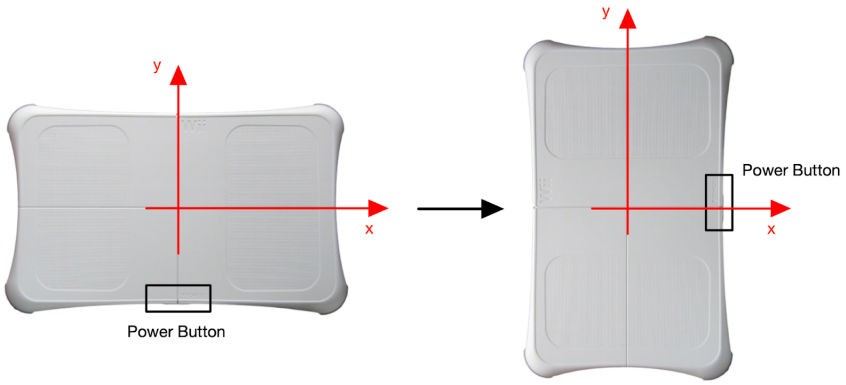

Figure 31: Reference system transformation

Our objective is to analyze the pattern of the Center of Pressure (CoP) signal during the squat activity. When employing a single balance board for both feet, it becomes unfeasible to effectively detect the CoP. As per its definition, the center of pressure is

inherently associated with a single foot only. Consequently, utilizing a single WBB and maintaining a state of perfect balance with both feet on it would result in the CoP position being registered somewhere between the two feet. Such a scenario aligns with the concept of balance rather than that of the Center of Pressure. Because of this, once we were able to connect and calibrate one balance, we worked on the code in order to make it easier to connect and save the data coming from two WBBs simultaneously. Basically we rotate the balance anticlockwise and we had to change the reference system accordingly with the convention mentioned in [13]. We changed the points coordinates of the calibration function called calibrationP. In fact, we maintained the same points but because of the new reference system, we changed their coordinates in [7, -15], [0,0] and [3, 6]. In general the main matlab script has been changed in order to connect and retrieve the data from two balance boards simultaneously. In the following lines we can explain in detail the main Matlab code.

```
1  plot = true;
2  wiiR = WiiBwCalibration('RIGHT', plot, 70, 0, path);
3  CalibrationModule(wiiR,0,0,0,'RIGHT', path)
4
5  wiiL = WiiBwCalibration('LEFT', plot, 70, 0, path);
6  CalibrationModule(wiiL,0,0,0,'LEFT', path)
7
8  Measure = 1;  % 1 if you want to measure, 0 if not
9  i = 1;
10 %% MEASURING
11 while Measure == 1
12     input('\n\nPress Enter to start measuring\n');
13     tic()
14     t=toc();
15     fprintf('Press A on 1st balance to stop measuring..    ');
16     m = 0;
17
18     starting_time = datestr(now, 'HH:MM:SS:FFF');
```

```
19      while ~isButtonPressedP(wiiR.bc.bb,'A')
20          cycleStart_sec = toc();
21
22          % query the wii for data
23          wiiR.update(plot);
24          wiiL.update(plot);
25
26          % when the Plot is off, this will help the user in order to see
27          % if the recording is going on
28          if m == 75
29              fprintf(' Recording...\n')
30              m = 0;
31          else
32              m = m + 1;
33          end
34          % pause before continuing cycle (subtracting the time
35          % taken to process this loop itself
36          cycleDur_sec = toc()-cycleStart_sec;
37          pause(1/wiiR.Fs - cycleDur_sec);
38
39      end
40      fprintf(' Measuring %d done\n\n', i)
41      toc()
42
43      % save data
44      wiiR.bc.saveAndClearAllData('RIGHT', false, path, starting_time);
45      wiiL.bc.saveAndClearAllData('LEFT', false, path, starting_time);
46
47      Measure = input('Enter 1 to start measuring again or 0 to stop: ');
48      i = i+1;
49  end
50  % clear memory
51  wiiL.bc.delete();
52  wiiR.bc.delete();
```

Code 3: Portion of WBB.m

This Matlab file called *WBB.m* represented in Code 3 is the main file. Most of the commands are repeated because they are for the two balance boards. On lines 2 and 5 there is the command *WiiBwCalibration* that is the initialization function for the class mentioned few sections above. This class takes the properties of the library WiiBalance plus the methods defined for calibrating the device. The first input of this function is the label that can be *'RIGHT'* or *'LEFT'*, then there is *plot* (true or false) for showing

the plots during the collection of the data. It was important to implement this option because enabling the plotting when more than one WBB is connected can really affect the sampling frequency of the software. The first input number represents the sampling frequency. The second number instead is the number of samples we neglect every second only for updating the plots. This option can be a trade-off in case it is important to show the plots but on the other hand it is necessary to reach a sufficient high sampling frequency. The last entry of this function is the path in the computer for accessing files with old calibration parameters or for saving performed measurements. In lines 3 and 6 there is the *CalibrationModule* function. This function has been defined for starting the calibrations. The first entry is the WBB of interest. The following three inputs can be only "0" or "1" where "1" is for running the calibration. The first "0" corresponds to the *NewtonTransform* calibration then we have *CalibrationWdrift* and *CalibrationP*. The user, therefore, can decide which calibration to perform. It is not necessary to perform all the calibrations every time.

Another important detail is in lines 23 and 24. The method *update* is the actual function that sample the data from the balance boards. Lines 44 and 45 are responsible for saving the collected data and lines 51 and 52 are important for clearing the memory and disconnecting the devices.

Chapter 4

IMPLEMENTATION - Control System Interface

The ankle exoskeleton discussed in Chapter 2.2 constitutes a rather intricate device. As previously delineated, The actuators are two motors controlled by an external PC programmed through a sophisticated Simulink program. The utilization of this device is by no means straightforward; a series of essential commands precede its operation, encompassing calibration and the implementation of systems to ensure a sufficient level of safety. In fact, the motors possess the capacity to exert a force that could impact the patient if not managed correctly. This sequence of sensor calibration commands, programming, and actuator enabling is entirely orchestrated by a Simulink program. This may be proved less intuitive for individuals unaccustomed to the use of complex software such as Matlab or Simulink. It is imperative to acknowledge that these technologies are intended for individuals lacking engineering or programming hard skills. Therefore, the utilization of an interface facilitating an intuitive initial setup was deemed crucial for device operation. Moreover, the operator must be able to receive feedback on the wearable robot's performance; hence, the user interface (UI)

must be capable of gathering necessary information from the device's sensors (such as encoders), processing it if necessary, and presenting it to the user.

The ankle foot exoskeleton is entirely managed by an external computer provided by Beckhoff Automation GmbH & Co. KG[1], a German company specializing in industrial automation and control technologies. The Simulink program operates on the external computer through Beckhoff TwinCAT software, offering a real-time PC-based automation environment. This TwinCAT software incorporates an additional tool called

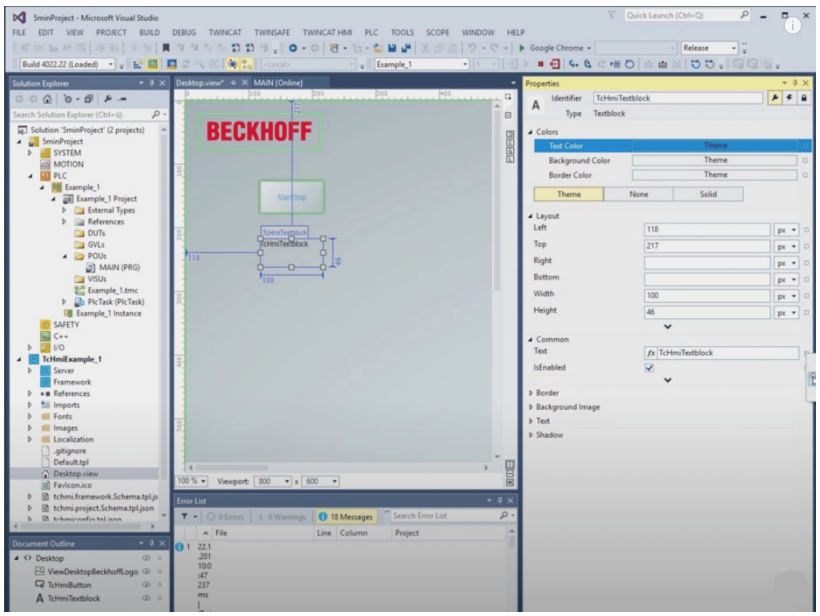

Figure 32: Twincat by Beckhoff

TwinCAT HMI (Figure 32), which constitutes a human-machine interface (HMI) system crafted to empower users to create graphical and intuitive interfaces for industrial automation and control systems. This software is employed for designing and displaying control screens, operator panels, and graphical interfaces, facilitating operators in

[1]https://www.beckhoff.com/en-us/

monitoring and controlling industrial processes to enhance interaction between operators and control systems. TwinCAT HMI typically supports various functionalities, including real-time data visualization, remote access and control, alarm management, and data logging. Its design allows flexibility and adaptability to the specific requirements of diverse industrial applications. Through this tool, it was possible to develop the interface depicted in Figure 33. The interface comprises three sections: the left

Figure 33: Control System User Interface

column contains functional flags to indicate potential errors or the current system status. The ESTOP button appears in red if the ESTOP switch is deactivated. Limit switch represents the state of the safety switch installed on the exoskeleton. If the angle becomes lower than -5 degrees it will change state and it disables the motors. Left and right motor error circles notify possible error messages coming from the motor drivers. The lowest row encompasses all the commands necessary for the user to calibrate and set the AFO. The first two buttons, labeled *Torque Tare* and *Angle Tare*, facilitate

encoders calibration to the patient's standing position. Subsequently, the *UDP/User* button alters the parameter input mode for the controller's ascending and descending gains. In its default state, the button sets the UDP mode, where new parameters are transmitted from the computer executing the optimization process. The operator observes an automatic update of the parameters in the first two boxes (filled with the numbers 15 and 40 in Figure 33). At this point, the user can confirm the update by clicking the *Descending_stiff* and *Ascending_stiff* buttons below. If the *UDP/User* button state is changed, the User mode is activated, requiring the operator to manually input parameters whenever changes are necessary, using the lower boxes. Further to the right, there is a dropdown menu enabling the selection of the controller mode. The configurations are No control, Position control, and Squat control. Finally, two switches allow the user to decide whether to save the data represented by the graphs and to enable the motors. The remaining central section contains six graphs divided into three columns: Angle, Torque, and Angle-Torque curve. Rows differentiate between left and right. For instance, the signal in the top-left graph represents the angle read by the left-side encoder. In the plots of the first column, there is also a discrete black signal representing the controller's changing of the state. In the central column's windows, two signals per graph are discernible. The red signal denotes the desired torque, while the blue signal represents the actual torque. Finally, the top-right graph displays the desired and actual angle-torque characteristics. The bottom-right graph, however, is not used.

In conclusion, we obtained an intuitive user interface designed for ease of operation. The application, incorporating safety flags, calibration controls, and graphical feedback, ensures user-friendly interaction and real-time monitoring. Overall, this system enables efficient and secure utilization of the ankle exoskeleton.

Chapter 5

IMPLEMENTATION - Optimization Interface

The main goal is to personalize the assistance of an ankle exoskeleton through incorporating user ECG feedback. The optimization algorithm (HIL toolkit) implemented in Python plays a crucial role in this scope, enabling the discovery of optimal solutions to assist the subject wearing the exoskeleton. However, the initial version of the optimization algorithm lacked a comprehensible graphical interface, hindering its accessibility for non-experts in the domain [1]. In this project, we address this limitation by developing a user-friendly application to enhance the configurability and real-time monitoring of the optimization process. The previous implementation required users to manually access the correct directory, access a YAML file and then modify it to change the initial settings, which proved cumbersome for non-technical individuals. Additionally, the absence of real-time progress tracking, pause functionality, and other features further restricted the algorithm's utility.

While the missing functionalities could have been incorporated by adding a few lines of Python code, we opted to create a dedicated application. This approach was

chosen to ensure that users, including those unfamiliar with programming languages, could intuitively interact with the optimization algorithm without requiring extensive technical knowledge.

The newly designed application introduces an intuitive graphical interface, simplifying the process of modifying initial configurations. Furthermore, it allows users to monitor the optimization process in real-time, offering pause and resume capabilities, and providing access to view other features of the optimization process.

In this chapter we will describe the design. In *Software structure* we will show all the Python scripts that we had to implement in order to make the structure reliable and efficient. It will also focus on the design of the *Server.py* file. In *GUI Design* we will talk more in detail about all the features of the web application.

5.1 Software structure

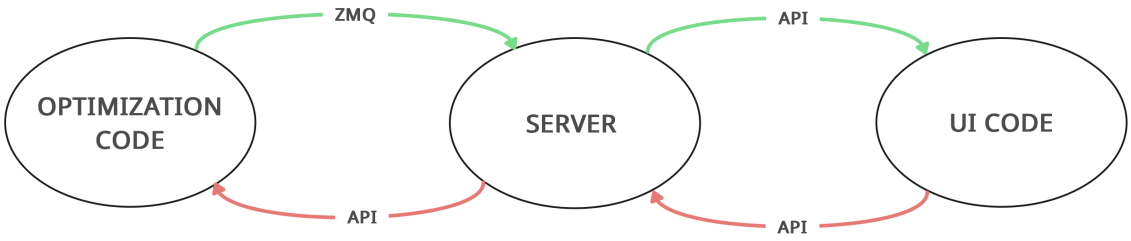

Figure 34: Software main architecture

In this part of the explanation we will often mention an *Optimization Code*, by this we mean the set of Python scripts that had been developed in order to run the Bayesian optimization algorithm. In other words, it is the HIL toolkit described in [1].

This code contains also the scripts needed to detect and retrieve the data from the ECG sensor. This optimization algorithm is responsible for creating all the outputs that we want to show in the UI. For *UI code* instead, we refer to all the Python scripts that are necessary to build the web application and to make working all the callbacks that it has within.

The goal was to design something that was independent from the optimization process. The rationale behind this decision is straightforward: it is possible to encounter errors during the execution of the Bayesian optimization, which may result in a terminal failure. In such a scenario, it is essential to prevent the application from failing altogether. Instead, we aim to enable the user to continue viewing the outputs and, if necessary, receive appropriate notifications regarding the possible errors the optimization process is undergoing. Moreover, it is imperative to ensure that any errors within the UI component do not in any way compromise the optimization process. It is important to bear in mind that the optimization procedure can span anywhere between 15 to 20 minutes approximately. Throughout this duration, the subject is required to perform squats, a physically demanding activity. Consequently, it is absolutely important that a simple UI error does not affect the optimization process, leading to the need for a complete restart of the entire experiment. Ensuring the separation of UI and optimization is essential in this context. The clear distinction between these components will promote a more reliable and resilient system, guaranteeing that both the user interface and the optimization algorithm can proceed harmoniously and without any undue interference.

For this reason, we decided to develop a third Python script called *Server.py*. This file is essential for the separation of the processes mentioned before. Its goal is to constantly listen to the optimization code in order to get new data, store it and share it whenever the UI calls for downloading the information. In Figure 34 we have green and red arrows. Green arrows represent all the data created by the optimization process and shared with the UI. The red arrows instead represent the command that the UI is able to send to the optimization code. These commands are "0" or "1", they stand for "PAUSE" and "RESUME". Moreover, on each arrow there is a label. That label says the protocol or mean through which the data is shared. We used the protocol ZMQ for sharing in an asynchronous manner the data of the optimization with the Server class. For the remaining connections we used a REST API.

5.1.1 REST API

REST API stands for Representational State Transfer Application Programming Interface and it is a set of rules on how to connect and communicate between client and server. In particular a REST API is an API that respects a specific architectural style. This style uses HTTP requests to access and use data. The methods that can be used by client/server are GET, PUT, POST and DELETE which refers to reading, updating, creating and deleting data on a specific endpoint of the server. The working principle is that clients can make a HTTP request described by one of the methods listed before. Then the server will respond with a status code. For example "200" means "Ok" and "404" means "Not found". In order to exploit this communication method

we imported the python library called *Requests*[1]. The Python library *Requests* is a widely used and essential tool for handling HTTP-based interactions within Python applications. It provides developers with a simple powerful interface to send various types of HTTP requests and effectively communicate with APIs. By abstracting away the underlying complexities of network communication, it significantly reduces the development time and effort required to implement HTTP functionalities within Python applications. Code 4 represents a portion of *Server.py*. This is an example on how to share the data when the client makes a request.

```
151 @app.get('/OptimizationData')
152 def list_OptimizationData():
153     return saved.share_data()
```

Code 4: Function for sharing data with UI

Whenever the client asks for a GET method on the endpoint '/OptimizationData' the Server returns to the client the outputs of the function *share_ data()*. This function simply gives all the data related to the optimization stored by the server in format json. JSON (JavaScript Object Notation) is a text-based data format. It is a set of key-value pairings, where the value can be a number, string, object, array, or boolean and the key must be of the string type.

Here instead we can see the Python command for the client in order to download the data from the server.

```
18 def download_data(obj, config):
```

[1]https://pypi.org/project/requests/

```
19      n_parm = config['Optimization']['n_parms']
20      server_flag = requests.get(f'http://{obj.serverIP}:{obj.serverPort}/
            OptimizationData')
21      if server_flag.status_code == 200:
22          obj.flags['server'] = 'ON'
23      else:
24          obj.flags['server'] = 'OFF'
25      data = server_flag.json()
```

Code 5: Downloading data from Server

In Code 5, the output of the command on line 20 is the status code of the response by the server. If the response corresponds to 'ok' then we can save all the data in line 25 using the method .json() of the library *Requests*.

5.1.2 ZMQ

Optimization code and Server use the ZMQ library in order to exchange the data. ZMQ[2] is a high-performance, open source asynchronous messaging library. It is built in C++ but there are several libraries translating this set of commands for other programming languages. ZMQ allows you to send messages (binary data, simple strings, objects) over the network through various methods like TCP (*Transmission Control Protocol*). The messages are exchanged through sockets. The sockets differ from type of message and properties. Types of socket are for example REQ/REP and PUB/-SUB. We decided to use the PUB/SUB type because, unlike REQ/REP, in this case the publisher pushes the messages out and all the associated subscribers receive the messages without sending responses or acknowledgments. After some tests, we understood it was the best method for exchanging our data. In order to use this library we

[2]https://pyzmq.readthedocs.io/en/latest/

created a class called *ServerCommunication*. In its initialization function there are all the steps to create the socket. In input we have to specify the host IP, the port and the communication type (if it is a socket for sending or receiving). These three inputs are defined in a YAML file called *ECG_ configs.yml* that is situated in the folder "configs" of the HIL toolkit. In Code 6 there are two examples of inputs needed for creating a socket. This is a portion of code of *ECG_ configs* related to the communication settings.

```
22 Communication:
23     Sending: true
24     RECEIVING: true
25     IP: tcp://192.168.1.49
26     HIL:
27         Cost samples:
28             NAME: Cost_samples
29             TYPE: send
30             PORT: 4501
31         FLAGS:
32             NAME: FLAGS
33             TYPE: send
34             PORT: 4507
```

Code 6: Setting up the ZMQ communication

As clear from Code 6, on line 23 and 24 we enable the communication for sending and receiving data. On the 25th row we set the IP of the device where we run the optimization. The IP is the address by which the device is recognized in the local network. Consequently, for each variable we define name, type and port. Every channel of communication has a different port. For example *cost_ samples* is the vector containing the sample of the cost function. We defined this variable as type “sending” and all the messages containing this variable will be exchanged on port 4501. In the class mentioned before, *ServerCommunication* we have listed also the methods that we

can use for all the sockets created.

```
22 def send (self, message: str) -> None:
23     self._socket.send_string(u=message)
24
25 def send_obj(self, obj: object) -> None:
26     self._socket.send_pyobj(obj)
27
28 def send_json (self, json: dict) -> None:
29     self._socket.send_json(json)
```

Code 7: ZMQ sending functions

Code 7 shows the three types of sending functions. Command on line 22 is for sending strings. Command on line 27 is for sending python objects like lists or dictionaries and command on line 31 is for sending messages in json format. We can show the same methods for receiving.

```
22 def receive (self -> str None:
23     try:
24         obj = self._socket.recv_string ()#flags=zmq.NOBLOCK
25     except zmq.error.Again:
26         obi = None
27     return obj
28
29 def receive_obj(self) -> object:
30     try:
31         obj = self._socket.recv_pyobj ()
32     except zmq.error.Again:
33         obj = None
34     return obj
35
36 def receive_ json(self) -> Any:
37     try:
38         obj = self ._socket.recv_json ()
39     except zmq.error.Again:
40         obj = None
41     return obj
42
43 def recv_bytes(self) -> bytes None:
44     try:
45         obj = self._socket.recv ()
```

```
46      except zmq.error.Again:
47          obj = None
48      return obj
```

Code 8: ZMQ receiving functions

We have similar commands for receiving ZMQ messages in the Server code.

5.1.3 Server Architecture

The Server file is divided in two sides: ZMQ and REST API side. The ZMQ side is the one responsible for receiving the data coming from the optimization code and the REST API side is connected to the UI code. For the first side, it is fundamental to be able to run many subprocesses in parallel. We have numerous variables, leading to a substantial number of channels for exchanging messages between Bayesian algorithm and UI. The conventional approach of using a single loop to sequentially receive messages from all the channels is proven to be inefficient. A potential issue arises when the sender transmits a message through a channel that is not actively being monitored at that moment due to our focus on another channel, resulting in information loss. To address this challenge, we propose employing Python's *Asyncio* library[3]. This powerful tool enables the execution of multiple subprocesses asynchronously and in parallel. By adopting this approach, we can concurrently run numerous subprocesses, with each subprocess dedicated to listen to a specific channel. Consequently, we can create subprocesses equal to the number of sockets or channels utilized for exchanging messages coming from the optimization code. Such an arrangement ensures that each subpro-

[3]https://docs.python.org/3/library/asyncio.html

cess remains continuously attentive to its designated channel, effectively eliminating the risk of missing any message. In the following figure we can see an example of an asynchronous process for the acquisition of ECG data.

```
87  async def data_ecg():
88      sock = ctx.socket (zmq.SUB)
89      sock.subscribe("")
90      sock.connect(f"{saved.IP}:{saved.port}09")
91      while True:
92          try:
93              msg = await sock.recv_json(flags=zmg.NOBLOCK)
94          except zmq.ZMQError:
95              msg = None
96          if msg is not None:
97              saved.ecg = msg
98          await asyncio.sleep(0.1)
```

Code 9: Server's subprocess for receiving ZMQ messages

We have seven subprocesses. Each of them looks like the one in Code 9 and it has the three commands that are shown in lines 88, 89, 90 for creating the socket and binding it with the sender at the specific address defined by IP and port. Then we have a While loop. In this loop we constantly try to receive a message, if we don't get anything the variable msg will be set to None. If we receive a message, instead, this will be saved in a local variable called *saved.ecg*. Because of this design with seven subprocesses, every time the system encounters an await function, like on line 93, it means that the system doesn't need to wait for that function to finish but it can move on with the other functions or subprocess.

The other side of *Server.py* is related to the Flask app, the REST API that provides the data to the UI. We mentioned the local variable *saved.ecg*. *saved* is an object of

the class *Store* that is defined in the beginning of *Server.py*. This class is important for initializing, storing and sharing all the variables with the web application. Code 10 shows two methods defined in the class *Store*.

```
16 def reset data(self):
17     self.plot = {'x': [], 'y': []}
18     self.gp = {'mean': [], 'x': [], 'y': []}
19     self.ecg = []
20     self.acq = None
21     self.hyp = {'likelihood.noise_covar.raw_noise': [],
22                 'mean_module.raw_constant': [],
23                 'covar module.raw_outputscale': [],
24                 'lengthscale parm1': [],
25                 'lengthscale parm2': [],
26                 }
27     self.state = "OFF" #state of the optimization process
28     self.hrv = None
29     self.opt_comand = "1" #command from UI for optimization
30
31 def share data self:
32     in_memory_datastore = {
33         "data plot": self.plot,
34         "data_gp": self.gp,
35         "data_acq": self.aca,
36         "data_hyp": self.hyp,
37         "data_ecg": self.ecg,
38         "data hrv": self.hrv,
39         "state": self.state}
40     return in_memory_datastore
```

Code 10: Methods used by Server code

The function *share_data* on line 31 is called by the client. Indeed the client makes a request with method GET on the endpoint '/OptimizationData' and the server will respond by giving the output of the function *share_data*. This connection between client requests and *share_data* is clear from the callback shown in Code 11.

```
151 @app.get('/OptimizationData')
152 def list_OptimizationData():
```

```
153     return saved.share_data()
```

Code 11: Callbacks for sharing data with UI code

As explained before, UI code can make different kinds of client requests. Here is Code 12 showing other callbacks in response to client's requests.

```
155 @app.get('/OptCommand')
156 def sendOptcommand ():
157     return saved.opt_comand
158
159 @app.post('/OptimizationData')
160 def reset_OptimizationData():
161     saved.reset_data()
162     return 'data reset'
163
164 @app.post('/OptResume')
165 def resume_opt():
166     saved.opt_comand = "1"
167     return 'comand received'
168
169 @app.post('/OptPause')
170 def pause_opt():
171     saved.opt_comand = "0"
172     return 'comand received'
```

Code 12: All callbacks used by Server code

Briefly the first callback is for the Optimization code. The HIL code can download the command "RESUME" or "PAUSE" set by the UI. The second callback is for clearing all the stored variables in the Server. The last two callbacks, resume_opt and pause_opt, are used by the UI to send the command of "RESUME" or "PAUSE" to the optimization process. Indeed with these two callbacks we basically change the local variable called saved.opt_comand and with the callback on line 155, the optimization code retrieves the value of saved.opt_comand from the Server.

5.2 GUI design

The design of this application is based on Dash, a python library[4]. Dash by Plotly is a dynamic web app framework that allows developers to create interactive web applications using Python. It uses Plotly's visualization features to transform data into user-friendly charts, and graphs. Dash's straightforward syntax integrates data manipulation, visualization, and user interaction, making it a versatile tool for creating intuitive web interfaces.

The Web application is constituted of two layers. The first layer is fixed and it contains the title of the application, two badges that give some information to the user and then we have a row that is a list of tabs. These tabs are Initialization, Optimization, Signals and Hyperparameters tab. This first layer occupies the top row of the screen and it is fixed, it's always present. The most important features of this layer are the badges. These badges are indicators of the status of Server code and Optimization code. Based on which color they get, they represent a different status. Table 1 describes the status for each color of the badges. For Server there are only

	RED	GREEN	ORANGE	BLUE
Server	Not working	Working	-	-
Optimization	Not started	Finished	Exploration	Optimization

Table 1: Legend for badges' colors

two colors because only two status are possible. The user has to know that if the Server badge is red that means that some error may occurred on the Server code or

[4]https://dash.plotly.com/minimal-app

the it hasn't been started yet. In this situation it is still possible to run the Bayesian algorithm but it won't be possible to see the outputs on the UI. For the optimization badge instead there are 4 states. Red when the optimization hasn't been started yet, orange when the optimization started and it is in the Exploration phase. Finally blue when the Bayesian algorithm is in the Optimization phase.

The second layer of this UI describes the remaining part of the screen and its configuration depends on which tab has been selected by the user. When UI code is executed and we open the application for the first time, the initial tab selected is the Initialization tab.

5.2.1 Initialization tab

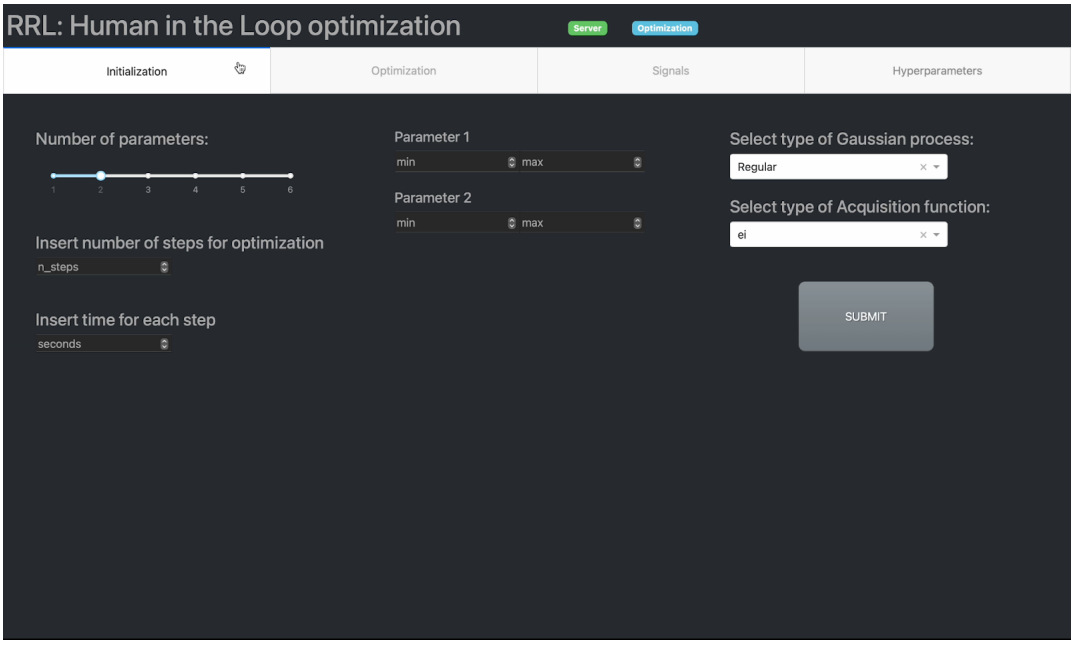

Figure 35: Initialization tab

In Figure 35 it is shown the appearance of the first tab called *Initialization*. This tab

is the interface through which the user is able to change the settings before starting the optimization algorithm. There are many features that can be set for the HIL toolkit, however only the most important settings have been displayed for the UI. The user can select the number of parameters to optimize with the slider in the top left part of the app. By changing the marker's position on the slider, we change the number of boxes in the central column of the screen. If the slider is set on 5 parameters for example, the central column will display 5 couples of boxes, one couple for each parameter. In this central column, the boxes on the left are for setting the minimum value the parameter can be and the right boxes are for setting the maximum. By taking a look at the bottom of the slider it is possible to notice two other boxes. The upper box is useful for determining how many steps and in particular samples, the user wants to analyze for the optimization. The second box determines how many seconds each step should last. Finally, the right column has two dropdown components. By clicking on the first dropdown we can select one of the available Gaussian process models. By clicking on the second it is possible to select the type of Acquisition function.

5.2.2 Optimization tab

Figure 36 illustrates the second tab titled *Optimization*. The primary purpose of this window is to present the evolution of the two main functions utilized in the Bayesian optimization algorithm: the Gaussian process and the Acquisition function. Furthermore, within this tab, users have the capability to manage the optimization process through the utilization of three buttons located at the bottom of the window.

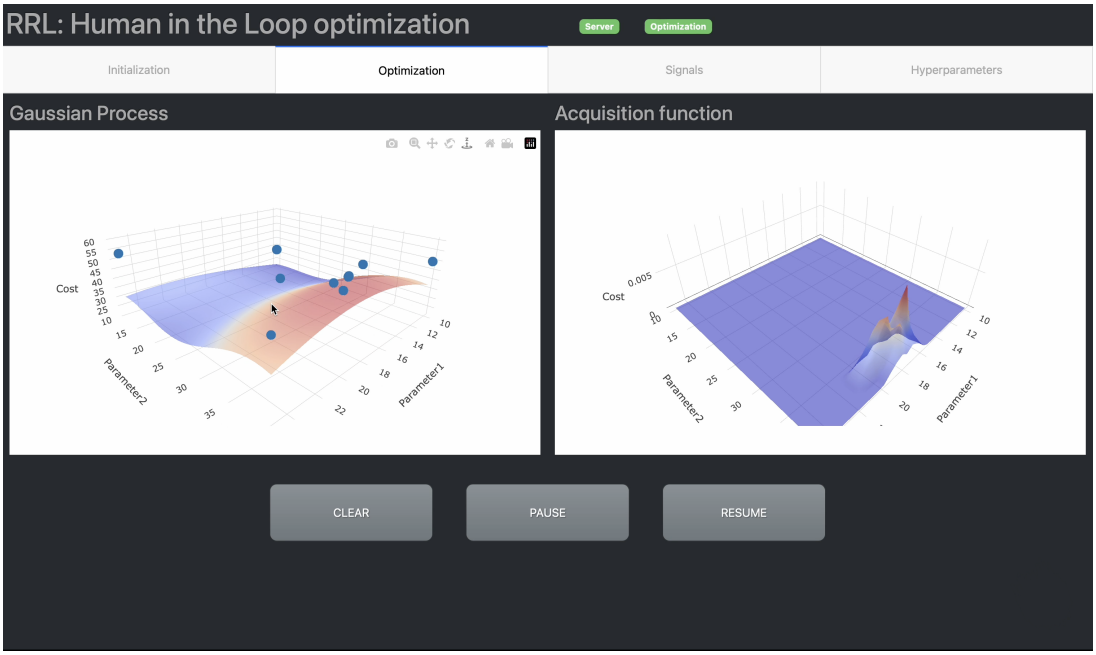

Figure 36: Optimization tab

On the left-hand side of the window, a graph is provided to plot the cost samples represented by blue points, which are utilized in the computation of the Gaussian process. Additionally, the graph displays the Gaussian process itself in the form of a colored surface. A useful feature of this graph is that by positioning the cursor on any of the blue points, a small dropdown is triggered, displaying the corresponding coordinates' values for that particular point. On the right-side of the window instead, it is possible to follow the evolution steps of the Acquisition function. Both the Gaussian process and the Acquisition function are interactive, as users can manipulate the orientation of the surface by dragging it. This enables users to observe the surfaces from various perspectives. However, it is worth noting that the application periodically updates the surfaces at fixed intervals of every three seconds, restoring their orientation to the

original position during each update. Three seconds should ensure that users have enough time to explore different perspectives before the update takes place. The three buttons available in this tab are CLEAR, PAUSE, and RESUME. When users activate the CLEAR button, the application initiates a POST request, prompting the client to request a reset on the server. Upon receiving this request, the server proceeds to delete all variables that were saved up to that point. Consequently, clicking on this button results in clearing all plots within the application, rendering them empty. As for the PAUSE and RESUME buttons, their functionality is explained in detail in section 5.1. These buttons allow users to halt and subsequently resume the optimization process as needed.

5.2.3 Signals tab

This section talks about the third tab labeled *Signals*. Here, we explore other essential signals related to the process. In the top left corner, we find the Biofeedback signal, which, for this project, is represented by the ECG signal used to compute the cost. The bottom left plot displays the trend of the RMSSD value, representing the cost computed using the ECG data. A slight difference exists between the RMSSD plot and the one in the bottom right of the screen. The plot on the bottom left illustrates the cost instant by instant, showing its variations over time. The optimization process samples this function to identify the optimal combination of parameters. All the sampled values from the cost function are shown in the bottom right plot. Lastly, the top-right graph demonstrates all the parameter values investigated during the exploration and

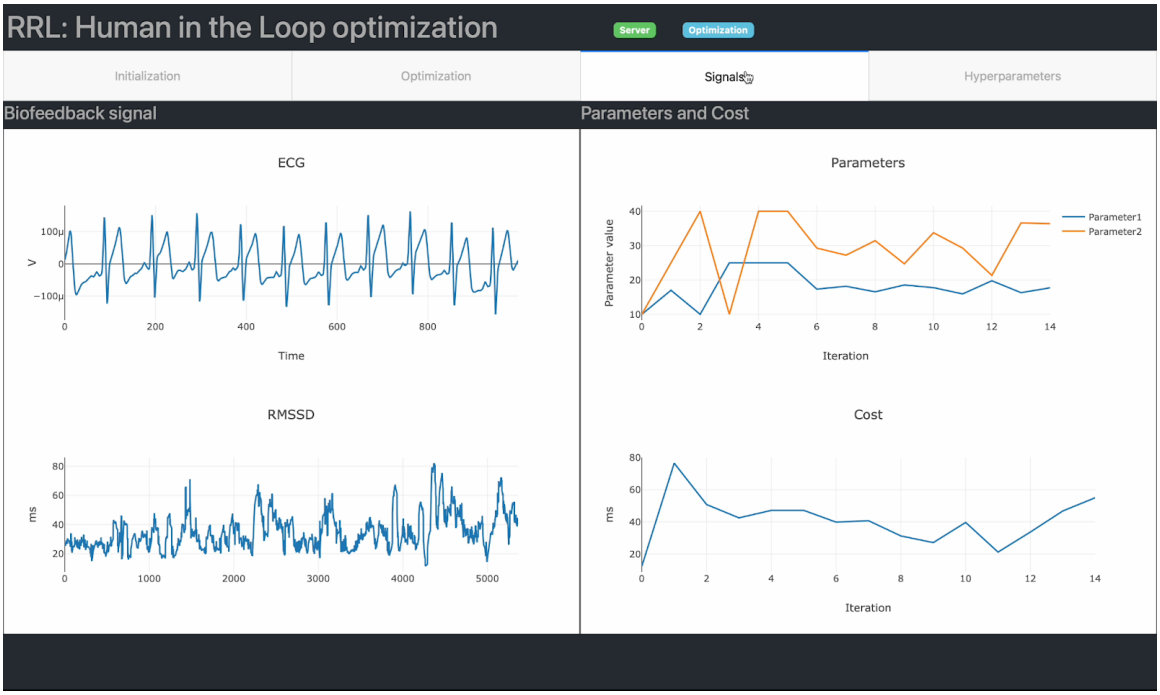

Figure 37: Signals tab

optimization phase. This graph allows the user to understand how distributed the exploration phase is or whether instead the optimization process found the optimal point more quickly.

5.2.4 Hyperparameters tab

The fourth tab is useful for evaluating the trend of the hyperparamenters. Hyperparameters are some parameters describing the Kernel used for calculating the Gaussian Process (GP) at each iteration. As already discussed in chapter 2.1 and in [1], these hyperparameters are very hard to set before starting the optimization. For this reason, a parallel machine-learning algorithm has been employed in the toolkit only for tuning the best value for the hyperparameters. Thanks to this tab, the users can evaluate the

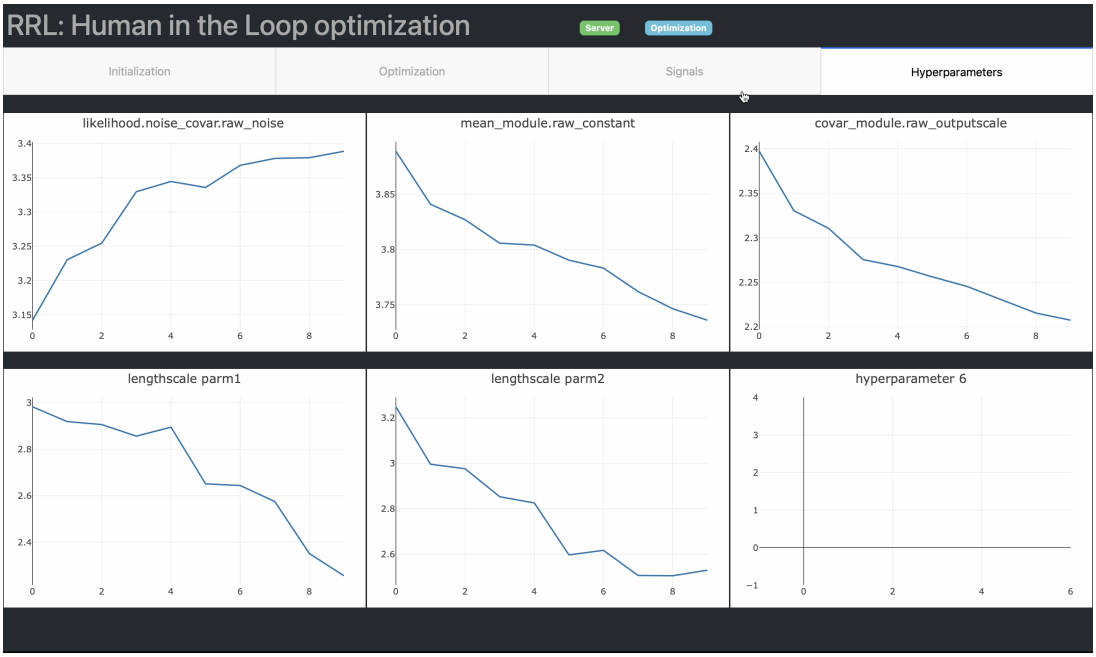

Figure 38: Hyperparameters tab

quality of the tuning of the hyperparameters. In fact, by looking at the plots showing their values over time, the user can check the convergence. If a hyperparameter converged to a certain value, there is a better chance of having achieved a good tuning of that hyperparameter. The number of hyperparameters depends on the number of parameters that we want to optimize for the wearable robot of interest. In particular the hyperparameters are as many as the optimized parameters plus three. In the example shown in Figure 34, we were testing optimizing two parameters that result in five hyperparameters to show. Indeed five graphs are filled out with the relative data and the sixth plot on the bottom right is empty. The idea behind this tab is to create as many graphs as number of hyperparameters. If there are more than six graphs, the user can scroll down this window and check all of them.

Chapter 6

IMPLEMENTATION - System Integration

In the field of wearable robotics, the concept of portability is of paramount importance. Currently, due to their complexity, limited practicality, and portability, these technologies are predominantly utilized within laboratory setups. Individuals with limited mobility can only take advantage of these technologies in specific locations, such as medical or rehabilitation facilities. For the advancement of this sector to be truly effective, it is imperative that individuals concerned have the opportunity to integrate this technology into their daily lives. This becomes particularly significant considering that a rehabilitation journey can span several weeks or even months. This reasoning led H. Jeong et al. to develop their exoskeleton with two compact actuators that can be accommodated, along with all other electronic components, within a backpack worn in conjunction with the Ankle-Foot Orthosis (AFO) [6]. The concept is to empower the user to harness the potential of the wearable device at any point during the day, whether at work or during a walk.

To achieve this goal, we committed ourselves to making all the discussed tools

as portable as possible, focusing on ease of use. Initial efforts were directed towards optimization. Optimization algorithms inherently require a computer for execution. Specifically in our case, a computer is essential for receiving sensor data providing biofeedback, processing it, and subsequently executing the optimization process. In line with our goal of maximizing portability, the decision was made to migrate the entire responsibility of running the Hardware-in-the-Loop (HIL) toolkit to a Raspberry Pi 3 Model B. The Raspberry Pi 3 Model B, developed by the Raspberry Pi Foundation, is a compact, credit-card-sized single-board computer designed for educational and hobbyist purposes, serving as a low-cost computing platform. It encompasses all essential components of a typical computer, including a processor, memory, input/output ports, and support for various peripherals. Operating on Linux-based systems, users can install diverse applications. With integrated Wi-Fi and Bluetooth, along with 4GB of RAM, this board offers extensive capabilities. Leveraging the potential of this device enabled the transfer of not only the data collection and computation aspects of the HIL toolkit but also the two essential scripts for running the Optimization UI. Given its diminutive size, this single-board computer is highly portable. Furthermore, the user does not require a cumbersome screen to monitor the activities of the exoskeleton (via the Beckhoff interface) or the HIL toolkit (via the optimization interface). These interfaces are accessible throughout the local network, making them available to any device capable of connecting to the same Wi-Fi network and equipped with a web browser. To facilitate this accessibility, we integrated a simple tablet into our portable setup.

The patient can easily search for the Raspberry Pi's IP address in the web browser and get access to the Optimization UI. Similarly, by entering the IP address of the computer running Twincat HMI (the laptop running the Simulink project), the user can access the Control System Interface. A more illustrative diagram is presented in Figure 39. The two green dashed lines in the figure represent wireless communications.

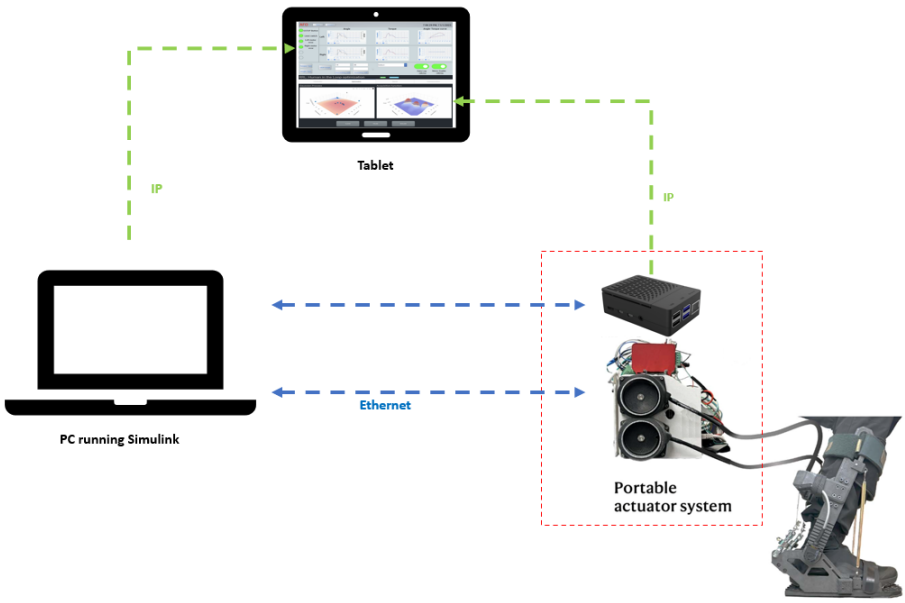

Figure 39: System Integration. Both interfaces are opened from the tablet.

Conversely, the blue lines represent wired Ethernet communications. The lowermost line makes the connection between the external PC and the PC running the Simulink control. The uppermost line, on the other hand, is a connection implemented to facilitate parameter updates. The toolkit has been slightly modified to share the new parameters through UDP communication protocol. Whenever a new set of parameters is generated during optimization, they are shared with the Beckhoff system, and the input boxes of the Control System Interface are updated. At that point, the operator

only needs to confirm the new inputs by clicking the appropriate buttons on the page open on the tablet (Figure 40). This integration of all systems makes the technology

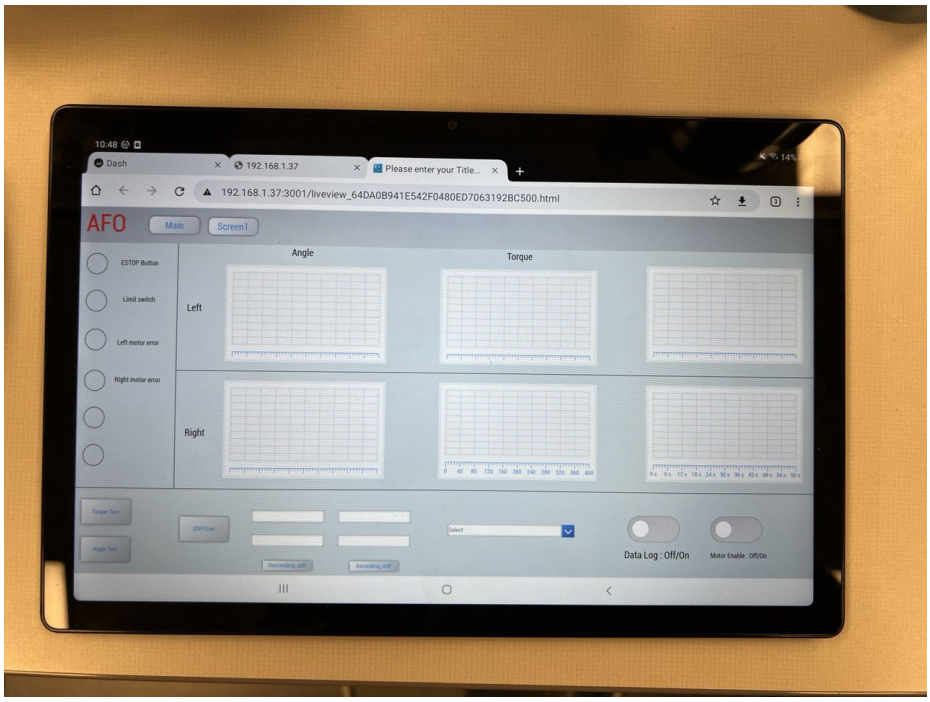

Figure 40: The Control System Interface opened on tablet.

potentially straightforward and independent of the need for support from others for its use. As explained in the introduction of this thesis, the achieved design is not yet sufficient to be truly considered portable and accessible to non-experts. The reasons are manifold. The control unit capable of independently managing commands for the exoskeleton is still executed in a bulky laptop. Another issue is power supply; the system is currently powered by a bench power supply. There are other bottlenecks as well. To use the Optimization UI, it is crucial to run the associated servers. In the current design, these scripts are executed on the Raspberry Pi (Single Board Computer SBC), requiring a monitor and keyboard to manually run these scripts. Access to the

Optimization app via the tablet is possible only locally. In other words, the application is not public, and the tablet must be connected to the same Wi-Fi network as the SBC to access them. In the proper functioning of this system, users should not be burdened with the task of initiating the servers to gain access to the interface, as it may prove to be a complex procedure for the majority of individuals. A viable solution involves employing an external computer (not necessarily connected to the same Wi-Fi network) dedicated to consistently running the scripts to keep the interface active. The single-board computer (Raspberry Pi) shares optimization data online with this external computer, allowing users to access the application simply by searching for the appropriate IP address in a web browser. However, it is important to note that the current technology is employed for experimental purposes, involving the processing of personal data, the privacy of which is paramount and must be protected. Sharing these data online to facilitate interface access from any location would pose a greater risk to privacy.

The focus on portability in wearable robotics is crucial for expanding beyond laboratory settings. Current limitations confine these technologies to specific locations, hindering daily use for individuals with limited mobility. The development of a portable exoskeleton represents a step towards integrating these technologies into users' daily lives. Our efforts, including migrating UI optimization scripts to a Single board computer and making User interfaces accessible through wireless devices, aim to enhance the accessibility and usability of wearable robotics. However, hardware challenges and

privacy concerns underscore the need for further refinements to achieve true portability and user-friendly integration.

Chapter 7

EXPERIMENT

To validate and assess the methodologies delineated in this thesis, an experiment involving a human subject was conducted. As elucidated in the introduction, the design of the two interfaces was iteratively formulated and tested incorporating the utilization of the Ankle-Foot Orthosis and Electrocardiogram sensor as biofeedback sensor. During the experimental phase, the interfaces developed in this thesis were employed in conjunction with a different optimization method, specifically a method based on Electromyography (EMG) sensors. The objective of this experiment was to demonstrate the effectiveness of EMG-based optimization in customizing the assistance provided by the exoskeleton during a squat activity. Operators leveraged the tools constructed and documented in this thesis, validating the systems (both interfaces and software for the Wii Balance Boards) and their integration. In addition to the system detailed in Chapter 6, EMG sensors, WBBs, and a respiratory mask (K5, Cosmed, Rome, Italy) were employed. The use of Wii Balance Boards was solely aimed at showcasing the functionality of the software for connecting and collecting CoP data. Conversely, EMG

sensors (Trigno, Delsys, MA, USA) served as biofeedback for the Human-In-the-Loop system. These surface sensors, with a sampling rate of 1259 Hz, were strategically positioned on specific points of both lower limbs following the SENIAM guidelines[17]. From these electrical signals, it became feasible to compute an average of the obtained signals and subsequently derive a cost function associated with the patient's physical effort. In terms of adapting the systems to this new feedback, the Human-In-the-Loop toolkit was inherently designed to accommodate various physiological signals. Similarly, the Optimization User Interface, owing to its versatility, seamlessly adapted to this feedback. Only the titles and labels of the graphs required modification, with no other alterations. The result of this adaptation is illustrated in Figure 41.

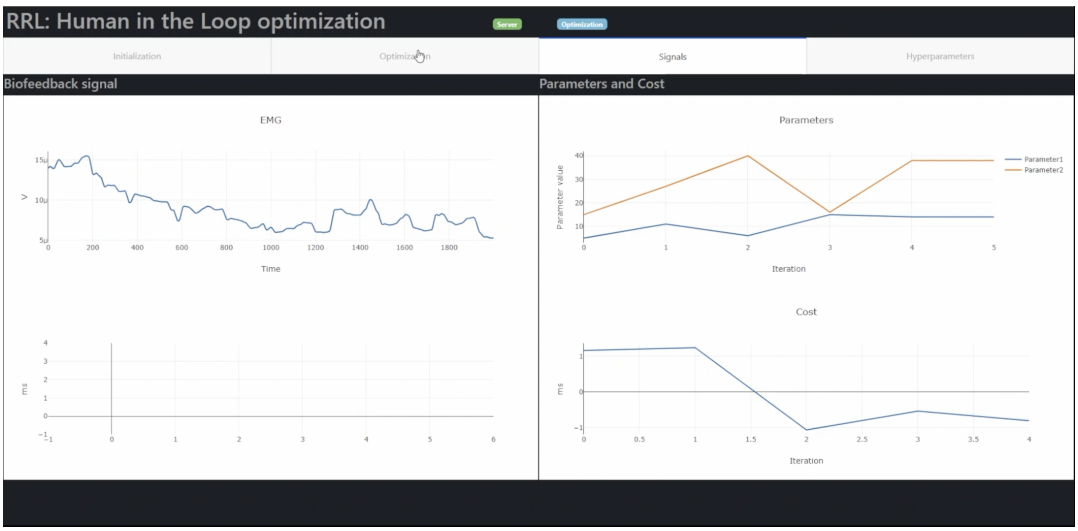

Figure 41: Signal tab from Optimization UI adapted to EMG optimization. Screenshot taken during the experiment.

The lower-left graph was not utilized. The plot in the top-left illustrates the mean voltage computed from the outputs of all EMG sensors. From this signal, the graph

in the lower-right is derived, representing the normalized cost for each iteration or set of parameters. The other tabs of the Optimization UI remained unchanged. For the WBBs, a separate laptop was employed. This latter device was utilized to execute the Matlab codes discussed in Chapter 3. Indeed, the initial step in commencing the experiment involved performing the calibration procedure for the WBBs (one for the left foot and one for the right foot). They were positioned to maintain an approximately similar distance to that between the subject's shoulders [18][19]. Following their placement and calibration, they were not relocated to avoid the necessity of re-calibrating the software.

7.1 The Protocol

The experimental protocol was adopted from [17] and received approval from the Institutional Review Board at the University of Illinois at Chicago, in accordance with the Declaration of Helsinki. A summarized schematic is presented in Figure 42. Primarily,

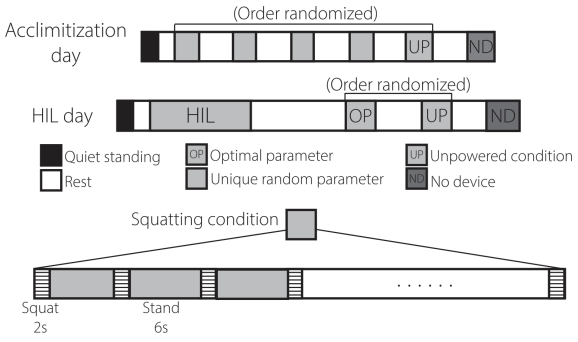

Figure 42: Experiment protocol. Figure from [17].

the experiment is divided into two days. The first day is termed the Acclimatization

Day, crucial for aiding the subject in acclimating to the device. On this day, four conditions are executed, during which the device is activated with random parameter sets. Following these four conditions, the Unpowered condition follows, where the AFO is worn but with deactivated motors, then there is the No Device condition, where squats are performed without the AFO. The exoskeleton is worn on the subject's dominant leg, while the non-dominant foot wears an insole to compensate for the height disparity caused by the device. During this acclimatization, only the Control System Interface is used, as no optimization is performed. On the second day, the Optimization Day or HIL Day, three total squat sessions occur. The first session involves optimization, utilizing the Optimization UI and lasting a maximum of 18 minutes, approximately corresponding to the execution of 105 squats. Subsequently, there is a rest period of 54 minutes, during which the subject rests in a seated position. The last three conditions follow: Optimal condition, where the exoskeleton is customized based on the HIL toolkit results, and finally, the Unpowered and No Device conditions. In all exertion sessions, squats must be executed in one second to descend and one second to ascend (totaling 2 seconds), followed by a 6-second standing position before restarting. In cases where a complete squat cannot be performed, the subject is instructed to go as low as possible while keeping their feet positioned at the center of the two WBBs. An operator monitors the subject's movements, providing verbal corrections in case of errors. A metronome marks the seconds, enabling the subject to follow squat and standing timings. The effort-to-recovery time ratio is 1:3. Hence, after the optimization

session, there is a 54-minute rest period, while after a set of 4 minutes, a 12-minute rest follows. This design allows the subject long time for complete recovery, rendering the fatigue measured by two distinct sessions independent.

7.2 The Results

This section is divided into two parts. The first part displays the interfaces and their behavior during the experiment. The second part presents the results through the Optimization interface, the data collected from the balance boards and the results of the experiment itself.

7.2.1 During Experiment

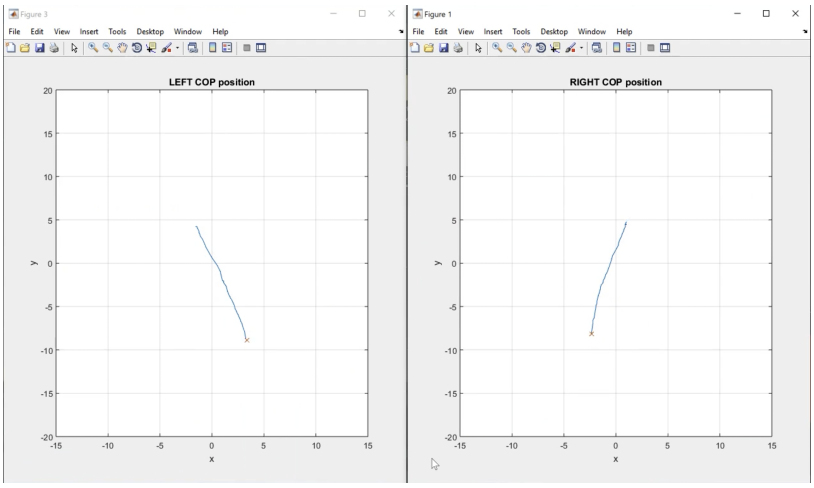

Figure 43: Screen of the laptop connected to WBBs. Screenshot taken during the experiment.

Figure 43 illustrates the screen of the laptop connected to the WBBs during the experiment. The user could observe the real-time movement of the right and left CoP.

At the end of the experiment, the user utilized the button on the WBB to halt data collection, and through the software, it was possible to save the collected data during the experiment.

As previously discussed in Chapter 5, the HIL toolkit and its corresponding UI operate in two phases known as Exploration and Optimization. The distinction between the two lies in the fact that in the former, the parameters for sampling the real cost function are already chosen by the operator and set in the code. In the latter phase, there is the Acquisition function, which, through exploration and exploitation methods, determines the optimal point at which to measure the real cost. The two phases of exploration and optimization are clearly distinguishable in the interface, as indicated by the color of the badge at the top, as depicted in Figures 44 and 45.

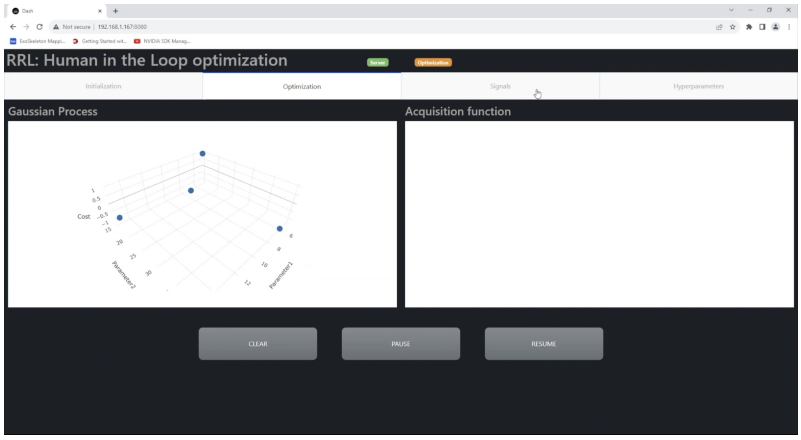

Figure 44: Optimization tab during exploration phase. Screenshot taken during the experiment.

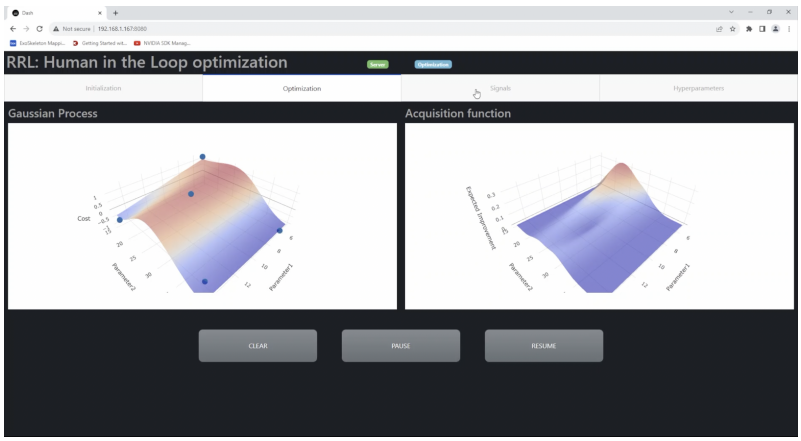

Figure 45: Optimization tab during optimization phase. Screenshot taken during the experiment.

As evident, the badge color undergoes a change. However, the more substantial distinction lies in the graphs. During the exploration phase (orange badge, Figure 44), only samples of the real cost function (depicted as blue dots) are available, the GP landscape is absent, and the Acquisition function is yet to be computed. Once the predetermined 5 points are measured, the optimization phase (blue badge, Figure 45) starts. In this phase, an initial approximation of the real cost function is established, and the Acquisition function is calculated on the right. The Hyperparameters tab serves a purpose solely in the Optimization phase. This is due to the absence of the GP cost function in the exploration, thereby precluding the tuning of Hyperparameters. Consequently, in the Exploration phase, the plots remain empty, while in the Optimization phase, they manifest as depicted in Figure 46.

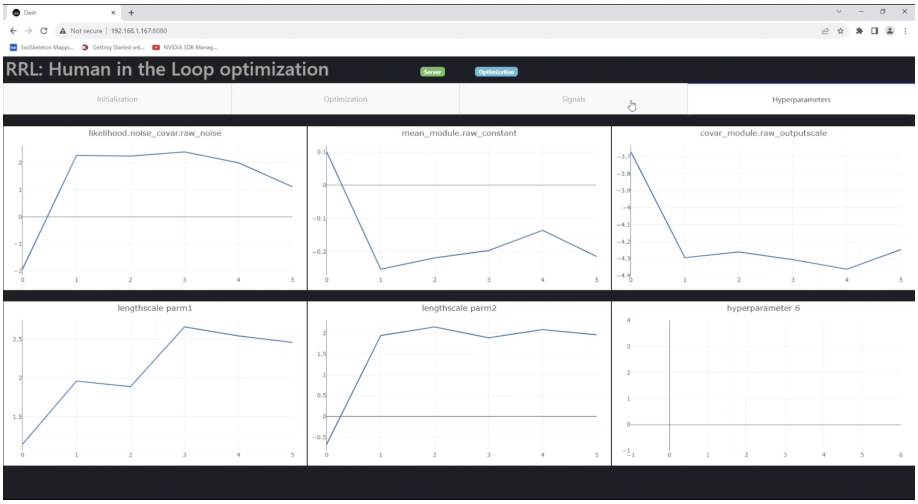

Figure 46: Hyperparameters tab. Screenshot taken during the experiment.

Transitioning to the Control System Interface, there are only two conceivable scenarios as it has been employed exclusively in the Unpowered condition (when the AFO is worn but inactive) and in the Optimization/Optimal condition (when the AFO is worn and active). In Figures 47, 48, and 49, various instances of exoskeleton application under different conditions are observable.

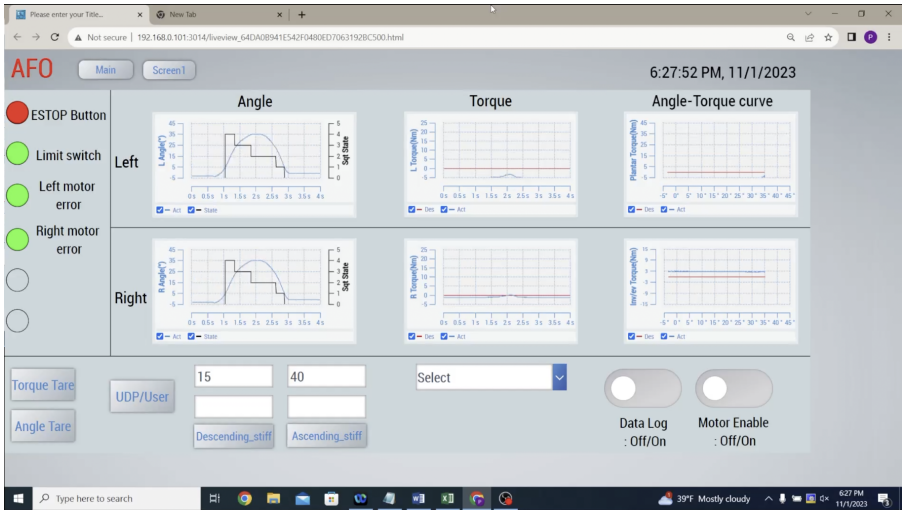

Figure 47: Control System Interface. Unpowered condition, desired torques and angle-torque characteristics are zero.

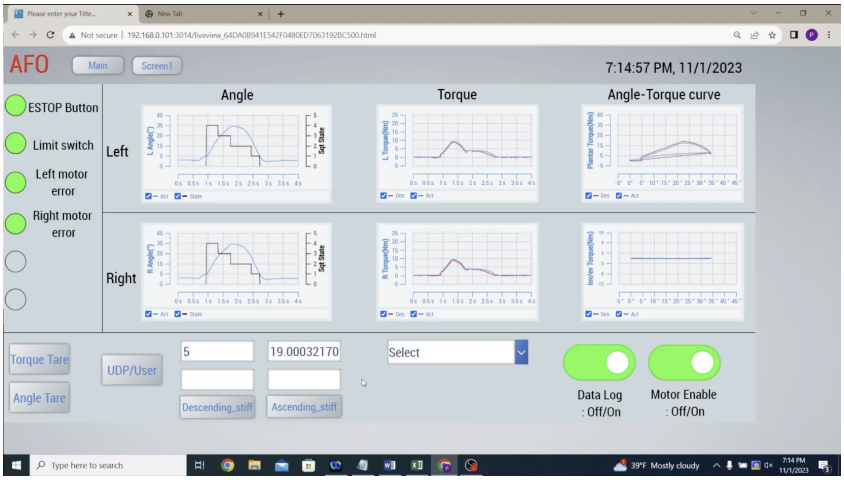

Figure 48: Control System Interface. Powered condition 1 with parameters 5 and 19.

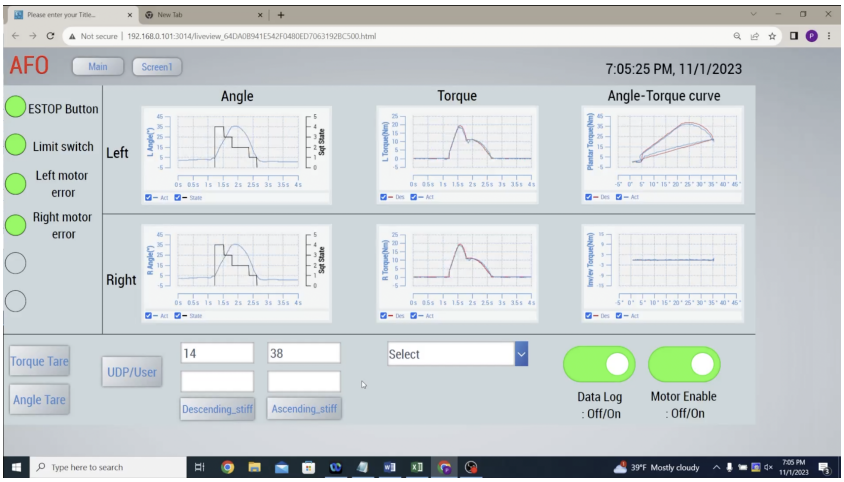

Figure 49: Control System Interface. Powered condition 2 with parameters 14 and 38.

7.2.2 After Experiment

In this section, we present the results obtained at the conclusion of the experiment. We commence by examining the data recorded from the two WBBs. Due to privacy considerations protected by the Institutional Review Board (IRB), it was not feasible to share here the experiment results. Consequently, to illustrate the functionality of the developed software, we conducted preliminary squats with the Wii Balance

Boards. Illustrative examples for the right foot are provided, Figure 50 shows the history movements of the CoP on the WBB surface. Figure 51 and 52 depict the x and y coordinates of the Center of Pressure over time. From Figure 50, it is discernible

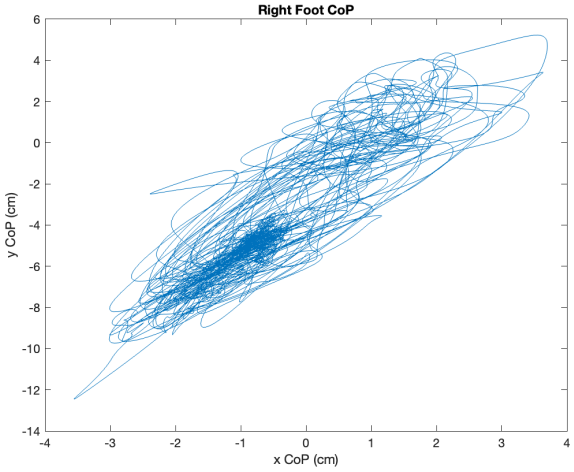

Figure 50: CoP movements during the squatting session.

that the equilibrium point during the standing position is located approximately at coordinates (-1, -6). The foot position on the balance board is notably open, elucidating the substantial variability in both signals (x and y). Properly placing the foot in a more vertical position could have resulted in appreciably more variable signals in the y-axis and less variability in the x-axis. When all the evaluations have been made and the optimization is done, the researcher can check the results. By keeping the web page open, the user can review the final outcomes (see Figure 53). At this point, the operator can employ the map to find out the parameter combination leading to the minimum effort. The user has the ability to drag the plot orientation, facilitating a more comprehensive view and analysis of the entire surface. Hovering the cursor over the surface provides precise cost values at specific points. Through an examination of

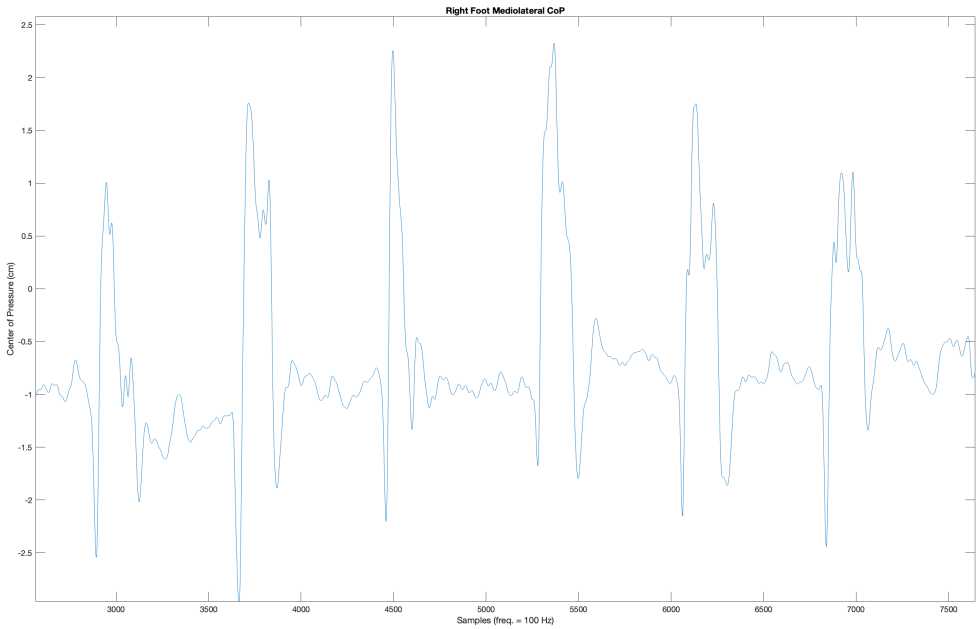

Figure 51: CoP x coordinate during 6 squats in time.

the surface in Figure 53, the experiment's researchers identified the optimal parameter combination, namely, 5 for descending (parameter 1) and 40 for ascending (parameter 2).

Following an extended rest period of nearly an hour, the subject was instructed to proceed with the final three conditions. The initial condition entailed the Unpowered condition, succeeded by the Optimal condition utilizing the aforementioned parameters, and concluding with the No device condition. In a post-processing analysis of the data, the researchers compared the normalized costs derived from EMG signals during the three conditions: The results are evident: the normalized cost of the Optimal condition is the lowest, registering at 2.56. It is followed by the Unpowered condition with a value of 2.72, and lastly, the No device condition exhibits the highest cost at 2.86. Consequently, the customization of the Ankle-Foot Orthosis resulted in a 5.9%

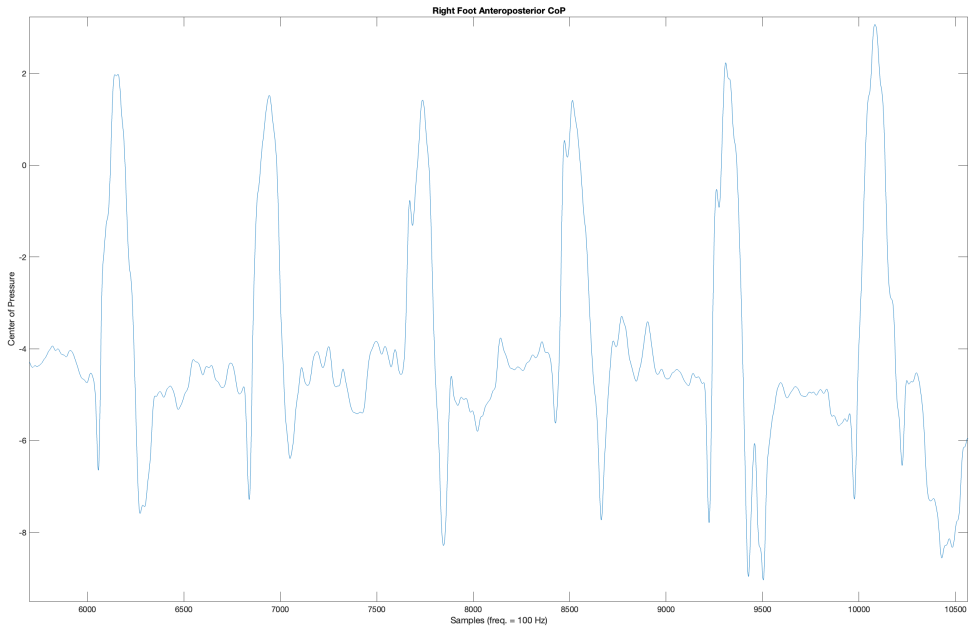

Figure 52: CoP y coordinate during 6 squats in time.

improvement over the Unpowered condition and a 10.4% enhancement compared to the No device condition.

The two interfaces haven't significantly contributed to the likelihood of achieving this outcome. However, they have concurrently enhanced the ease of managing and monitoring the experiment itself.

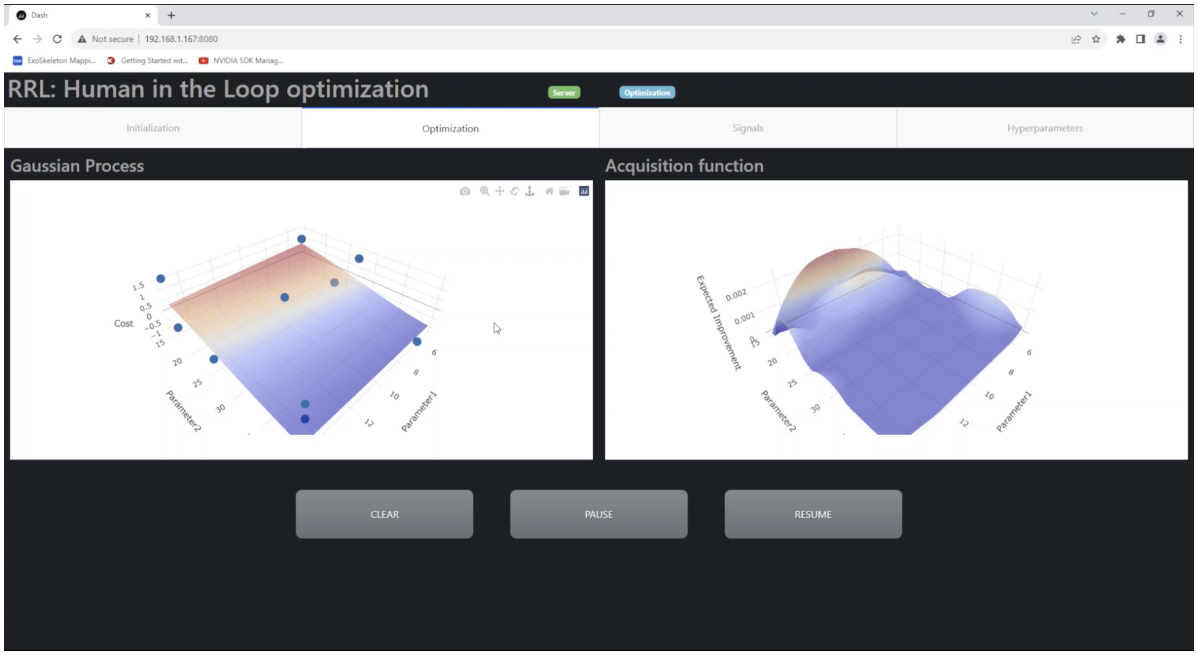

Figure 53: Final Gaussian process and Acquisition function.

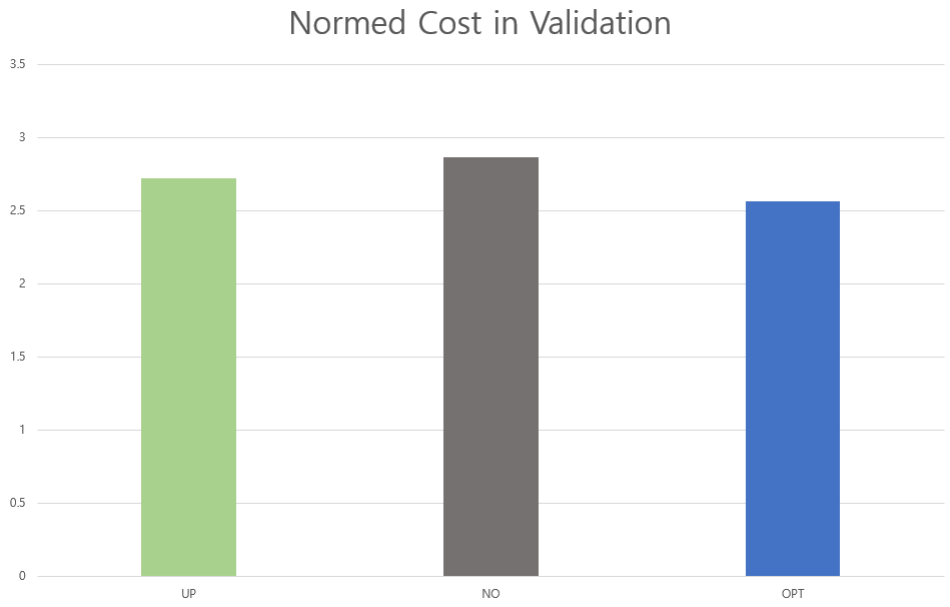

Figure 54: Comparative analysis of costs across the three squatting conditions

Chapter 8

CONCLUSION

In the course of this endeavor, the objective was to design tools to take a step towards simplification and enhanced portability of Wearable Robotics technologies. The Rehabilitation Robotics Laboratory had already made significant progress toward this goal, exemplified by the portable exoskeleton and the HIL toolkit. This thesis succeeded in developing two tailored interfaces to maximize the functionality of these systems in the simplest manner possible. For the AFO, a dedicated commercial software was employed to create human-machine interfaces in industrial applications or automated systems. Notably, the development of this interface consumed considerably less time than the second, the Optimization UI, which was programmed in Python using the Plotly Dash library. Additionally, this application required some modifications to the HIL toolkit itself to efficiently collaborate with the toolkit, resulting in an interface capable not only of displaying optimization process outputs but also of sending inputs, thereby allowing users to customize initial settings and decide to pause, resume, or reset the process. Regarding the Wii Balance Boards, they represent a cost-effective tool

that could unlock incredible research opportunities. We undertook the task of taking an existing library, enhancing it by implementing various features for calibration, data visualization, and storage. The program was also modified to enable the simultaneous connection of multiple WBBs. Here, the aim was to simplify its use, and although a User Interface is not present, the program prints messages in the terminal during its execution, guiding users through various procedures. The experiment documented in this thesis aimed to support another research project (EMG-based optimization of the AFO). However, researchers from this other project were able to leverage and benefit from the tools described in this work. The experiment yielded positive results, and the interfaces facilitated the management of the experiment itself, demonstrating versatility, simplicity, and effectiveness. These achieved results represent only a preliminary step towards the true potential these interfaces can offer. One immediate improvement concerns the software for the Wii Balance Boards. If their data is to be used in real-time as a physiological signal for the HIL toolkit, translating this code into Python would be necessary. Since Python has a library (PyLSL, Lab Streaming Layer) that allows the optimization process to effortlessly acquire all necessary data, the current software can only be useful for post-collection data analysis and cannot easily communicate in real-time with the HIL toolkit. As for the Control System Interface, there are still too many limitations to make the system truly portable. A possible improvement could involve migrating the computational power and the Simulink controller from the laptop to a single-board computer or a PC capable of entirely managing the AFO con-

troller while being portable and be incorporated into the Portable Actuator System. In this case, the interface would be created again from scratch since Twincat software wouldn't be accessible anymore. Turning to the Optimization UI, there are solutions that could be implemented immediately. The most crucial is improving its portability. The codes for opt UI currently managed by the Single Board Computer (Raspberry Pi) could be migrated to a laboratory computer. This computer executes the codes to keep the interface associated with the optimization process active. This system would ensure that the interface servers are always active, making the UI accessible at any time without requiring a non-expert to manually execute Python codes to start the application. In this case, however, and unlike the current system, the application should be public, accessible even from devices outside the local network. Currently, the tools have been utilized in laboratory environments and for experimental purposes, emphasizing the protection of personal data of subjects participating in experiments. However, in the future, consideration could be given to developing a more complex interface with an initial login section where the user must enter a username and password to access personal data. Another improvement could involve adding a tab to configure the physiological signal. Depending on the inserted type of feedback, the Signals tab adjusts the names of the graphs and their units of measurement. In conclusion, the tools developed and discussed in this thesis represent a possible solution for the portability and accessibility in Wearable Robotics, paving the way for further enhancements and applications. These tools have demonstrated their utility in experimental

settings, and as technology continues to evolve, the potential for broader applications and improvements remains promising.

CITED LITERATURE

[1] Prakyath Kantharaju et al. "Framework for Personalizing Wearable Devices Using Real-Time Physiological Measures". In: *IEEE* (2023).

[2] Jeong Hyeongkeun et al. "Muscle coordination and recruitment during squat assistance using a robotic ankle–foot exoskeleton". In: *Sci. Rep.* (2023).

[3] H. Han et al. "Selection of muscle-activity-based cost function in human-in-the-loop optimization of multi-gait ankle exoskeleton assistance". In: *IEEE* (2021).

[4] P. Slade et al. "Personalizing exoskeleton assistance while walking in the real world". In: *Nature* (2022).

[5] K.A. Ingraham, E.J. Rouse, and C.D. Remy. "Accelerating the estimation of metabolic cost using signal derivatives: Implications for optimization and evaluation of wearable robots". In: *IEEE Robot. Autom. Mag.* (2020).

[6] Inigo Sanz-Pena, Hyeongkeun Jeong, and Myunghee Kim. "Personalized Wearable Ankle Robot Using Modular Additive Manufacturing Design". In: *IEEE Robotics and Automation Letters* (2023).

[7] Fred Shaffer, Rollin McCraty, and Christopher Zerr L. "A healthy heart is not a metronome: An integrative review of the heart's anatomy and heart rate variability". In: *Front. Psychol.* (2014).

[8] Michael Scott, Graham Kenneth S., and Davis Glen M. "Cardiac autonomic responses during exercise and post-exercise recovery using heart rate variability and systolic time intervals—A review". In: *Front. Physiol.* (2017).

[9] Laurent Mourot et al. "Quantitative Poincaré plot analysis of heart rate variability: effect of endurance training". In: *Eur. J. Appl. Physiol.* (2004).

[10] M.P. Tulppo et al. “Quantitative beat-to-beat analysis of heart rate dynamics during exercise”. In: *Amer. J. Physiol.-Heart Circulatory Physiol.* (1996).

[11] Jihoon Kim et al. “Soft wearable flexible bioelectronics integrated with an ankle-foot exoskeleton for estimation of metabolic costs and physical effort”. In: *Nature Portfolio Journal* (2023).

[12] Dominque Makowski et al. “NeuroKit2: A Python toolbox for neurophysiological signal processing”. In: *Behavior Research Methods* (2021).

[13] Fabio Scoppa et al. “Clinical stabilometry standardization: Basic definitions – Acquisition interval – Sampling frequency”. In: *Gait & Posture* (2012).

[14] Lena H. Ting, Harrison L. Bartlett, and Jeffrey T. Bingham. “Accuracy of force and center of pressure measures of the Wii Balance Board”. In: *National Institute of Health* (2014).

[15] Julia M. Leach et al. “Validating and Calibrating the Nintendo Wii Balance Board to Derive Reliable Center of Pressure Measures”. In: *Sensors* (2014).

[16] Jordan Brindza et al. “WiiLab: Bringing Together the Nintendo Wiimote and MATLAB”. In: *IEEE* (2009).

[17] Prakyath Kantharaju et al. “Reducing Squat Physical Effort Using Personalized Assistance From an Ankle Exoskeleton”. In: *EMB* (2022).

[18] R. F. Escamilla et al. “A three-dimensional biomechanical analysis of the squat during varying stance widths”. In: *Med. Sci. Sports Exerc.* (2001).

[19] Adam M. Kushner et al. “The Back Squat Part 2: Targeted Training Techniques to Correct Functional Deficits and Technical Factors that Limit Performance”. In: *Strength Conditioning J.* (2015).

VITA

NAME	Gregory Sacco
EDUCATION	Master of Science in Electrical and Computer Engineering, University of Illinois at Chicago, Dec 2023, USA Master Degree in Mechatronics Engineering, Dec 2023, Polythechnic of Turin, Italy Bachelor's Degree in Electronics Engineering, Oct 2021, Polythechnic of Turin, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency