ESCUELA DE
INGENIERÍA DE FUENLABRADA

DEGREE IN BIOMEDICAL ENGINEERING

End of Degree Project

SIGNAL PROCESSING IN TENDON REFLEX MEASUREMENTS: SOFTWARE DEVELOPMENT AND IMPLEMENTATION

Author: Raquel Perea Calvo

Supervisor: Antonio R. Consoli Barone

Supervisor: Francisco Suarez Castro

Academic Course 2023/2024

“I never once failed at making a light bulb. I just found out 999 ways not to make one."

— Thomas A. Edison

Acknowledgements

At the end of my degree in Biomedical Engineering, I realize that on graduation day, I will be the only one wearing the band and receiving my diploma. However, without all the people who have supported me and encouraged me to continue throughout these years, I would never have come this far. I may not be able to add your names to the diploma, but what I can do is thank you with all my heart and dedicate a few words to you in this paper.

First and foremost, I extend my sincere thanks to my tutors, Antonio R. Consoli Barone and Francisco Suarez Castro, for their guidance, insightful feedback, and encouragement.

I am deeply grateful to my parents. My father, who has always jumped at the opportunity to help me whenever I asked, who has always tried to spark my curiosity to discover and understand what initially seemed impossible, and who has motivated me to follow the path of engineering. To my mother, who with great patience has helped me get back up every time I fell, who has dried my tears every time I cried, who has never let me give up, an who has been a true role model and an incredible woman to look up to. Their love, encouragement and support, have made this journey possible, and I cannot thank them enough for all the sacrifices they have made on my behalf.

I would like to dedicate a special mention to my grandparents, who have always believed in me, and whose wisdom and kindness have motivated me to strive for excellence. To my grandmother Conchi, who I know would be so proud.

I would also like to thank all my college classmates, with whom it has been a pleasure to share these past years. Especially to my friends María and Julia, who have always been there to listen, understand, and offer their help and friendship whenever needed. Their support and the moments of laughter we shared provided balance and relief during the challenging times.

I would like to thank a very special person who has been by my side since the day we met, always present no matter the distance, a mile or a thousand miles away. Thank you for cheering me up even on the worst days, for always making me smile, and for never giving up on me.

Lastly, I would like to mention the institution making this project possible, thanks to the Rey Juan Carlos University and all the people I have encountered during my years studying here, all the skills and knowledge they provided make the engineer I am today.

To all of you, I extend my heartfelt thanks. This achievement would not have been possible without your unwavering support and encouragement.

Abstract

Reflexes are involuntary responses produced by our organism in response to stimuli, triggered by the activation of medullary or brainstem circuits, crucial for the survival of species.

Somatic reflexes are routinely assessed by physicians to evaluate the proper functioning of the nervous system. A commonly tested reflex is the patellar reflex, elicited by tapping a reflex hammer onto the patellar tendon, due to its easily observable response. While this traditional test is widely performed, innovative devices equipped with a sensor-integrated reflex hammer and electromyography (EMG) can record reflex response activity and facilitate calculations such as latency measurement.

The challenge with these advanced devices and their associated software lies in the lack of automation for latency calculation. The primary objective of this project is to address the needs of the URJC laboratory staff by developing an intuitive and reliable tool capable of automating these calculations, enhancing efficiency and accuracy in the assessment of reflex responses.

To meet the objectives set for this project, a graphical user interface (GUI) was developed, providing users with the ability to select specific files, visualize signals, process them according to their needs, and calculate latencies. The GUI includes a graphing toolbar that allows users to amplify and navigate through the signals for detailed inspection. The program offers multiple filtering options, including low-pass, high-pass, and band-pass filters, as well as a Savitzky-Golay filter. Various latency calculation methods are available, enabling users to calculate latencies based on the maxima of the reflex hammer signal, and the maxima, minima, absolute value, or derivatives' maxima of the EMG signal.

Results were acquired testing the different filters and latency calculation methods to determine their characteristics, so the user can later decide which processing and post-processing methods suit their needs and goals better. To grant an easy and fast access to users, this program is implemented as an .exe file, allowing a seamless integration into existing workflows.

Recognizing the potential for continuous enhancement, this project includes a discussion on future improvements and possibilities. The open-source nature of this project facilitates these advancements, inviting further contributions and refinements.

The primary objectives were successfully achieved, resulting in the development of a reliable and intuitive tool for automated latency calculations. This tool reduces potential human error, decreases time expenditure, and lowers research costs, thereby enhancing overall efficiency and accuracy.

Resumen

Los reflejos son respuestas involuntarias producidas por nuestro organismo en respuesta a estímulos, desencadenadas por la activación de circuitos medulares o del tronco encefálico, cruciales para la supervivencia de las especies. Los reflejos somáticos son evaluados rutinariamente por los médicos para evaluar el adecuado funcionamiento del sistema nervioso. Un reflejo comúnmente evaluado es el reflejo patelar, desencadenado al golpear con un martillo de reflejos el tendón rotuliano, debido a su respuesta fácilmente observable. Si bien esta prueba tradicional se realiza ampliamente, existen dispositivos innovadores equipados con un martillo de reflejos integrado con sensor y electromiografía (EMG) pueden registrar la actividad de la respuesta a los reflejos y facilitar cálculos, como la medición de la latencia.

El desafío con estos dispositivos avanzados y su software asociado, radica en la falta de automatización para el cálculo de la latencia. El objetivo principal de este proyecto es abordar las necesidades del personal del laboratorio de la URJC desarrollando una herramienta intuitiva y confiable capaz de automatizar estos cálculos, mejorando la eficiencia y precisión en la evaluación de las respuestas a los reflejos.

Para cumplir con los objetivos establecidos, se desarrolló una interfaz gráfica de usuario (GUI), que proporciona a los usuarios la capacidad de seleccionar archivos específicos, visualizar señales, procesarlas según sus necesidades y calcular latencias. La GUI incluye una barra de herramientas gráficas que permite a los usuarios ampliar y navegar a través de las señales para una inspección detallada. El programa ofrece múltiples opciones de filtrado, incluidos filtros de paso bajo, paso alto y paso banda, así como un filtro de Savitzky-Golay. Se dispone de varios métodos de cálculo de latencia, lo que permite a los usuarios calcular latencias basadas en los máximos de la señal del martillo de reflejos, y los máximos, mínimos, mayor magnitud o máximos del gradiente de la señal de EMG.

Se obtuvieron resultados al probar los diferentes filtros y métodos de cálculo de latencia para determinar sus características, para que el usuario pueda decidir más tarde qué métodos de procesamiento y postprocesamiento se ajustan mejor a sus necesidades y objetivos. Para brindar un acceso fácil y rápido a los usuarios, este programa se implementa como un archivo .exe, lo que permite una integración perfecta en flujos de trabajo existentes.

Reconociendo el potencial para una mejora continua, este proyecto incluye una discusión sobre mejoras y posibilidades futuras. La naturaleza de código abierto de este proyecto facilita estos avances, invitando a contribuciones y mejoras adicionales.

Los objetivos principales se lograron con éxito, lo que resultó en el desarrollo de una her-

ramienta confiable e intuitiva para cálculos automatizados de latencia. Esta herramienta reduce el error humano potencial, disminuye el gasto de tiempo y reduce los costos de investigación, mejorando así la eficiencia y precisión en el análisis de datos.

Contents

List of Figures

List of Acronyms and Abbreviations

ACh Acetylcholine

AP Action Potential

DC Direct Current

EEG Electroencephalography

EMG Electromyogram

FIR Finite Impulse Response

GUI Graphical User Interface

HD-sEMG High Density Electromyogram

LCA Latency Calculator Application

MOL Muscle Onset Latency

MU Motor Unit

MUAP Motor Unit Action Potential

NINDS Neurological Disorders and Stroke

URJC Universidad Rey Juan Carlos

Chapter 1

Introduction

1.1 Context and motivation

Reflexes are involuntary responses that our organism produces in the presence of a stimulus. These stimuli may vary, including mechanical, luminous, electrical, or thermal, among others. However, the response emitted by the nervous system is combined by a chemical and an electrical process [1].

Reflexes constitute standardized reactions to a given stimulus, triggered by the activation of medullary or brainstem circuits, without requiring the cerebral cortex or other superior centres to process that information.

Reflexes are normally checked by physicians to assess the correct functioning of the nervous system, and they are normally evaluated in the clinical practice as their alteration, either in their absence or in their hyperactivity is a sign of the underlying neural circuitry malfunctioning. Nevertheless, because there is electrical activity produced by the muscle, an electromyogram (EMG) can be used to record and measure this activity. The EMG testing along with a nerve conduction study is performed normally when muscle or nerve malfunctioning is suspected [2].

EMG devices may also be used for investigation, and they may include a hammer sensor or a grip force sensor. The goals of the experiments are diverse, and researchers need a broad spectrum of algorithms to analyze the data they extract.

In the case of this project, an EMG and a reflex hammer equipped with a sensor are both used. This kind of equipment is not often used clinically but it is useful in academic and research centres. The device is from AD Instruments Ltd, and both the EMG and the hammer come as

part of the same set, called PowerLab15T [3].

This project arises from the need to automate the calculation of the patellar reflex latency – the time the body takes to respond to a stimulus – given by the reflex hammer strike on the patellar tendon. The reflex hammer and the EMG signal are captured in two different channels, so latency calculations were previously performed manually using tools such as spreadsheet softwares or other commercial programs, such as LabChart, the one provided by the AD Instruments company.

Although there are other resources for similar tasks, none of them addressed our specific problem: automatically calculating latency for different types of reflexes, with varying stimuli nature, and employing different methods for recording these stimulus-reflex circuits. Additionally, commercial softwares can cost between 10000€ and 15000€, significantly increasing research expenses.

Furthermore, clinicians and researchers are not necessarily proficient in programming. Thus, creating a free, intuitive, and reliable tool can help reduce human error, save time, and lower overall research costs. Moreover, we aimed to develop an adaptable tool that can be modified as users' needs evolve, extending the software's useful life and reaching a wider range of potential users.

1.2 Objectives

The main objective of this project is to design and implement a software that allows acquisition, processing and analysis of data measured in tendon reflex measurements.

The data consists of two temporal traces: a stimulus trace (given by the electronic reflex hammer) and a muscular response trace (an EMG signal from the patient's muscular response). Utilities and functions provided by the software will provide data import, visualization and analysis tools in a simple and intuitive manner for the user.

In order to achieve the primary goal, the following secondary objectives were fixed:

- *Secondary objective 1. Development of a Graphical User Interface (GUI)* for visualization and analysis of measured data.
- *Secondary objective 2. Import Data.* Provide the GUI with an upload function which allows the user to import the selected files (in *.txt* format).
- *Secondary objective 3. Visualization.* Provide the GUI with a visualization tool for a clear representation of the imported data, capable of zooming in, zooming out and selecting the desired time frame of the measured traces.
- *Secondary objective 4. Processing of the signals.* Provide the GUI with specific and electable filtering options of the imported data.
- *Secondary objective 5. Latency calculations.* Provide the GUI with latency calculation functions, i.e. find maxima and calculate delays from the measured data. Latency calculations can be applied to the entire temporal sequence or selected portions of it. Results will be saved in a output file (*.txt* format).
- *Secondary objective 6. Executable.* Implement the software program as a stand alone application that can be executed on a Windows platform (*.exe* file).

These objectives as a whole serve to create a tool that enhances the field of biomedical engineering, offering greater accessibility and integration, serving as a bridge between physiology and computer science.

Chapter 2

State of the art

2.1 Reflexes

There are involuntary responses produced by our nervous system, which play a crucial role in our continued existence as a species. The responses to external stimuli which result in a rapid movement reaction are called somatic reflexes, the patellar reflex is an example of this kind of reflex. Somatic reflexes are nearly instantaneous movements because they bypass the brain; instead, they reach the spinal cord or brainstem, which sends the impulses back to the muscle that is going to perform the action. This shorter neural pathway allows for quicker reactions, improving our ability to respond to external stimuli.

2.1.1 Historical context

The study of reflexes has a very rich and diverse history. It took humankind centuries before reaching the understanding we have today of the concept of reflexes. Their study can be traced back to notable Greek scholars, such as Aristotle and Galen, who were the first to theorize on the nature of these involuntary movements. They stated that reflexes were not governed entirely by the soul, thereby paving the way for future investigations in the centuries to come.

Moving forward in time, during the Renaissance, the French erudite René Descartes was able to further develop the concept of reflex. Despite his rudimentary understanding of human anatomy, he observed that in human behavior, there were movements that were being performed unconsciously. For example, Descartes noticed that even when it was a friend blowing into his eyes, he would unconsciously close them. Therefore, the movement originated from a more

primitive action and was not controlled by the soul.

Other scientists throughout the years continued to test reflexes by experimentation, elaborating on the concept, until Charles Sherrington, at the end of the 19th and beginning of the 20th century, discovered, thanks to his physiological investigations, that reflexes were the fundamental units of the nervous system. This led to the foundation of our understanding of reflex arcs and the neural pathways involved in these automatic responses.

After the mechanism of reflexes was understood, the interest of 20th-century scientists turned, and was now focused on the discovery of new reflexes, like the muscle stretch reflex or the patellar reflex. But more importantly, these reflexes could be an important step towards the assessment of the well-being of the nervous system. The incorrect functioning of a patient's reflexes could be the indication of neurological damage or nervous system malfunctioning [4].

2.1.2 Reflex mechanism

Homeostasis

Reflexes play a major role in the preservation of homeostasis in our organism, comprising a combination of physiological processes that maintain bodily features such as pH, temperature, or blood sugar, among others. Homeostasis could be explained as the ability of the body to maintain stability within the internal environment, while the external circumstances and surroundings change.

Homeostasis is arguably one of the most important physiological concepts, crucial in the evolution and well-being of organisms. When homeostasis fails, the body is led to a pathological state that will only cease once the body reaches self-regulation equilibrium once again. Over the years, since the ancient Greeks to modern medicine, treatments have been oriented towards restoring balance within the organism.

The most recent addition to the understanding of homeostasis is the explanation of the networks through which balance is maintained. These processes function thanks to positive and negative feedback systems, which constantly monitor the environment and serve as backups in case a mechanism fails.

A feedback system is a closed loop that has four main components: the variables that need to be measured, a sensor that assesses the exterior and interior environment, and the variable in question, a processing unit that compares the values that the sensor is sending, processes that information, and creates an output that is sent back to the system, and the effector that acts if

the change of the variable requires it.

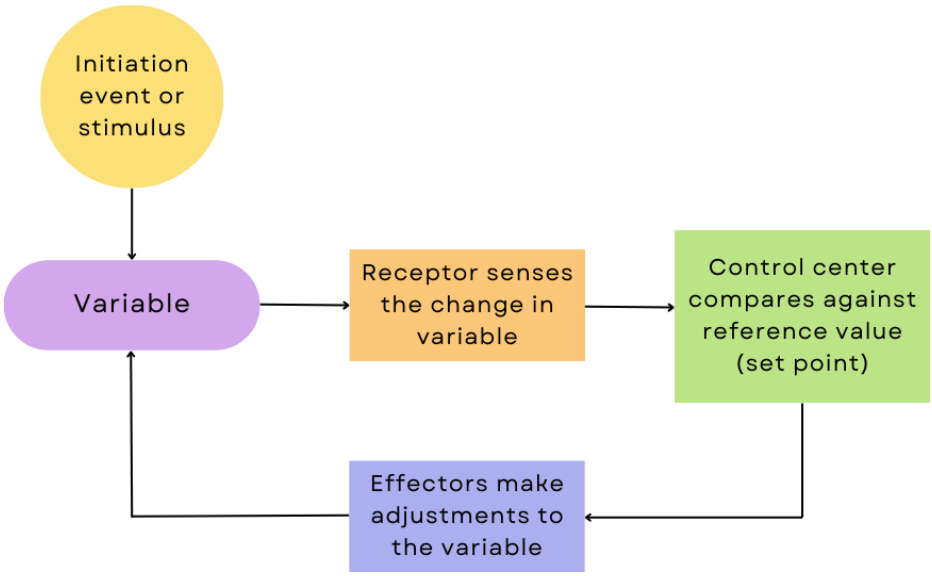

Figure 2.1: The schematic process of a negative feedback.

A positive feedback system, after receiving an input, produces an output that follows the same alteration as the input. In other words, a positive feedback loop would be one that, if the input is an increase in temperature, the output would also increase the temperature. Positive feedback mechanisms destabilize a given physiological system and commonly, they need a mechanism to stop it; nevertheless positive feedbacks are very useful to amplify biochemical or electrochemical signals within living organisms.

On the other hand, a negative feedback system counteracts the action of the input. For example, when the temperature rises, a negative feedback would decrease the temperature as a response. This is a stable system, and it is very useful when maintaining balance and equilibrium.

Homeostasis is sustained by both negative and positive feedback systems, with negative feedback being the most common one, as it maintains balance by counteracting actions such as

the rise in temperature in the body, which the negative feedback system counteracts by producing sweat. However, positive feedback systems also play an important role in some processes, such as blood clotting [5].

One of the main homeostatic processes is the postural maintenance and unconscious processes, like digestion, which are controlled by the somatic and autonomic reflexes, explained in Section 2.1.3 [6].

Reflex arc

One of the great utilities of reflexes is postural maintenance. In the motor system, there are negative feedback mechanisms that keep muscle length within an optimal range for force production, where opposite muscles counteract to maintain posture, which is ultimately another result of homeostasis.

A reflex arc is a neural pathway that enables the correct functioning of reflexes. The arc is composed of two pathways: the afferent pathway, which goes from the receptor to the integration center, and the efferent pathway, which goes from the integration center to the effector. The simplest reflex arc is going to need only two neurons: a sensory and a motor neuron. Reflexes that only need these two neurons are called monosynaptic reflexes. However, in other more complex reflexes (polysynaptic), interneurons are involved in the process [6].

In order to understand the mechanism of the reflex arc, it is necessary to understand the following concepts:

- The sensory receptor: This perceives the stimulus, which can be mechanical pressure, temperature, or light, among others.
- The sensory neuron: Part of the afferent pathway, it connects the sensory receptor to the integrating center.
- The integrating center: This is where the response to the stimulus is generated. It can happen in both, the spinal cord and brainstem.
- The motor neuron: Part of the efferent pathway, it connects the integration center with the effector.
- The effector: These are the muscles and organs that carry out the response.

The reflex arc starts when there is a stimulus noticed by the body, specifically the sensory receptor. Then, the sensory neuron undergoes an action potential, through which the stimulus reaches the integration center, the gray matter of the spinal cord. In the integration center, an appropriate response is generated and sent through a motor neuron to the effector organ, which carries out the response.

This is the process conducted by our body when, for example, we get poked by a needle. The sensory receptors in the skin, designed to identify pain, called nociceptors, detect the painful stimulus caused by the needle entering the skin. The information is then sent by the sensory neuron as electrical signals to the gray matter in the spinal cord, where the decision to move away the finger is made. Then, signals are sent back through motor neurons to the muscles that are going to take part in the response, the effector muscles [7].

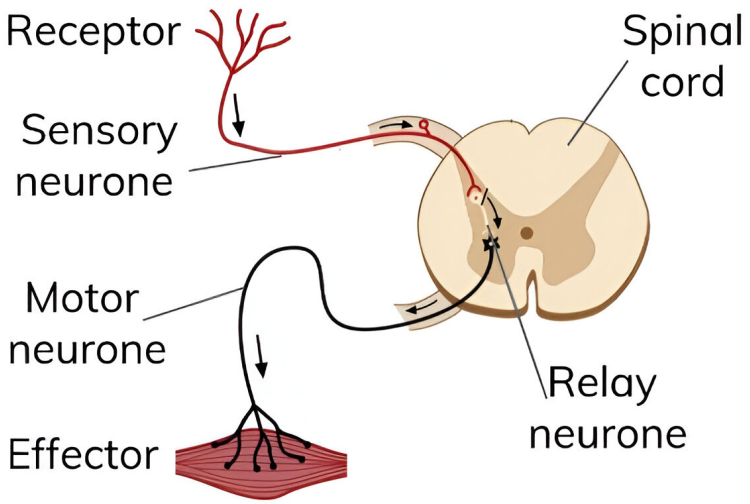

Figure 2.2: Illustration of the reflex arc mechanism [8].

2.1.3 Reflex classification

There are many ways to classify reflexes depending on the features to address. Some of the most significant classifications are, according to its function:

1. **Somatic reflexes:** They are responsible for maintaining posture and weight support in the body. They are controlled by the somatic nervous system, and the effectors for these types

of reflexes are skeletal muscles, which can be consciously moved if desired. Proprioception plays a major role in the maintenance of posture, sending feedback to the nervous system about the position and movement of the body. The main afferent proprioceptors for these kinds of reflexes are the Golgi tendon organs and the muscle spindles, which send information back to the spinal cord about muscle tension and length, respectively. Some examples of somatic reflexes are the patellar reflex or the withdrawal reflex [9] [10].

2. **Autonomic reflexes:** Controlled by the autonomic nervous system, they are responsible for unconscious processes, such as digestion or heartbeat. The effectors in this type of reflex are cardiac muscle, smooth muscle, and some glands. Some examples of these reflexes are the light response of the retina, or the maintenance of blood pressure [6].

Based on the location of the receptor organ:

1. **Deep tendon reflexes:** Also known as myotatic reflexes. The receptors for this kind of reflex are integrated within the muscle, and it is called the muscle spindle. The muscle spindle detects the tapping on the tendon, which will start the reflex arc process, resulting in the contraction and extension of a group of muscles acting on the response sent by the spinal cord. Examples of these types of reflexes are the plantar or the Achilles reflex.

2. **Superficial reflexes:** Since the receptor in this type of reflex is integrated within the skin, a smooth caress activates the reflex process. An example of this type of reflex is the abdominal reflex, which is generated by lightly stroking the skin in the abdominal area, resulting in the contraction of the abdominal muscles [11] [12].

Based on the location of the arc reflex:

1. **Spinal reflexes:** These reflexes are the ones where the integration system is the spinal cord, meaning that the reflex does not reach the brain. These reflexes are crucial when assessing damage to the spinal cord due to different factors. First of all, they can provide information about the location of the injury, because the sensory neurons innervate different heights of the spinal cord, making it possible to test which ones reach a healthy integrating system and which do not. It is also important when assessing the evolution of the injury; if there is an improvement in reflexes after the spinal shock, it can provide doctors with information about the severity of the injury. Some examples of these reflexes are the patellar or the plantar reflex [13].

2. **Cranial reflexes:** In the case of these reflexes, the reflex arc happens in the brainstem instead of the spinal cord. Visual reflexes, neuro-vegetative reflexes, or blood pressure are only some examples of the functions these reflexes fulfill. A cranial reflex to highlight because of its importance in assessing brain damage, is the pupillary light reflex. This reflex occurs when a bright light is shone into the eyes, causing the pupil to constrict in response to light. It occurs whether the person is conscious or unconscious. If the pupils are fixed and dilated, the patient is likely suffering from brain death, making the testing of this reflex a priority examination when there has been trauma to the head, due to its quickness and simplicity [14] [15].

Primitive reflexes

Primitive reflexes are reflexes developed prenatally and which persist for the first months or years of a child's life. These reflexes are essential for the correct adjustment of the newborn to the environment. The most important functions of these reflexes are the correct feeding of the newborn and protective reflexes against abrupt changes in their bodies. As the brain matures, these reflexes start disappearing or changing.

A very characteristic primitive reflex is the palmar grasp, which is provoked by pressing the palm of the baby with an object or finger, resulting in the baby's hand closing around the object. Another main infant reflex is the sucking reflex, which is induced by touching the baby's lips, causing the newborn to start sucking, thereby encouraging proper feeding [16] [17].

Both the absence of these primitive reflexes in newborns and their reappearance in adulthood may be indicative of brain or nerve damage. An example of the latter is the Babinski sign, which in infants is tested as the plantar grasp. When there is a suspicion of neurological damage in adults, a battery of tests is performed, one of them being the Babinski maneuver.

This procedure is performed by firmly rubbing a stick on the side of the sole of the foot. In the absence of injury, the big toe should perform a downward movement as if the foot were closing. If the big toe performs an upward movement, it could indicate pathology. When this maneuver is performed on a baby, an upward movement of the toe is expected and completely healthy. That is why this is a great example of how positive testing for primitive reflexes in adults can explain neurological damage [18].

2.1.4 Patellar reflex

The patellar reflex is classified as a deep tendon, monosynaptic, spinal, and somatic reflex. To evaluate this reflex, it is necessary for the patient to be in a sitting position with their legs hanging without touching the floor, or with the patient lying down, with both the hip and the knee of the same leg partially bent while the doctor places their hand beneath the knee to stretch the tendon. Once the posture is correct, the physician will find the patellar ligament and will proceed to tap it to induce the reflex.

After a sensor located in the quadriceps detects the hammer strike as a stimulus, the reflex begins. From this sensor, the signal travels through a sensory neuron to the spinal cord, where the segmental innervation is between the L2 and L4 vertebrae. Because it is a monosynaptic reflex, no interneurons intervene in the transmission of the response, which goes directly through the motor neuron to the quadriceps femoris muscle, which contracts as the effector [12].

Muscle spindles detect the stretching of the muscle that occurs when the tendon is struck. This helps us appreciate the position and movement of our body while adjusting the flexion and extension of opposing muscle forces and maintaining posture. When a joint straightens and the muscle fibers stretch as a result, specific spindles in that muscle are activated, enabling nerve fibers within the muscle spindles to detect how a joint is moving, how fast it is moving, and its position, among others [19].

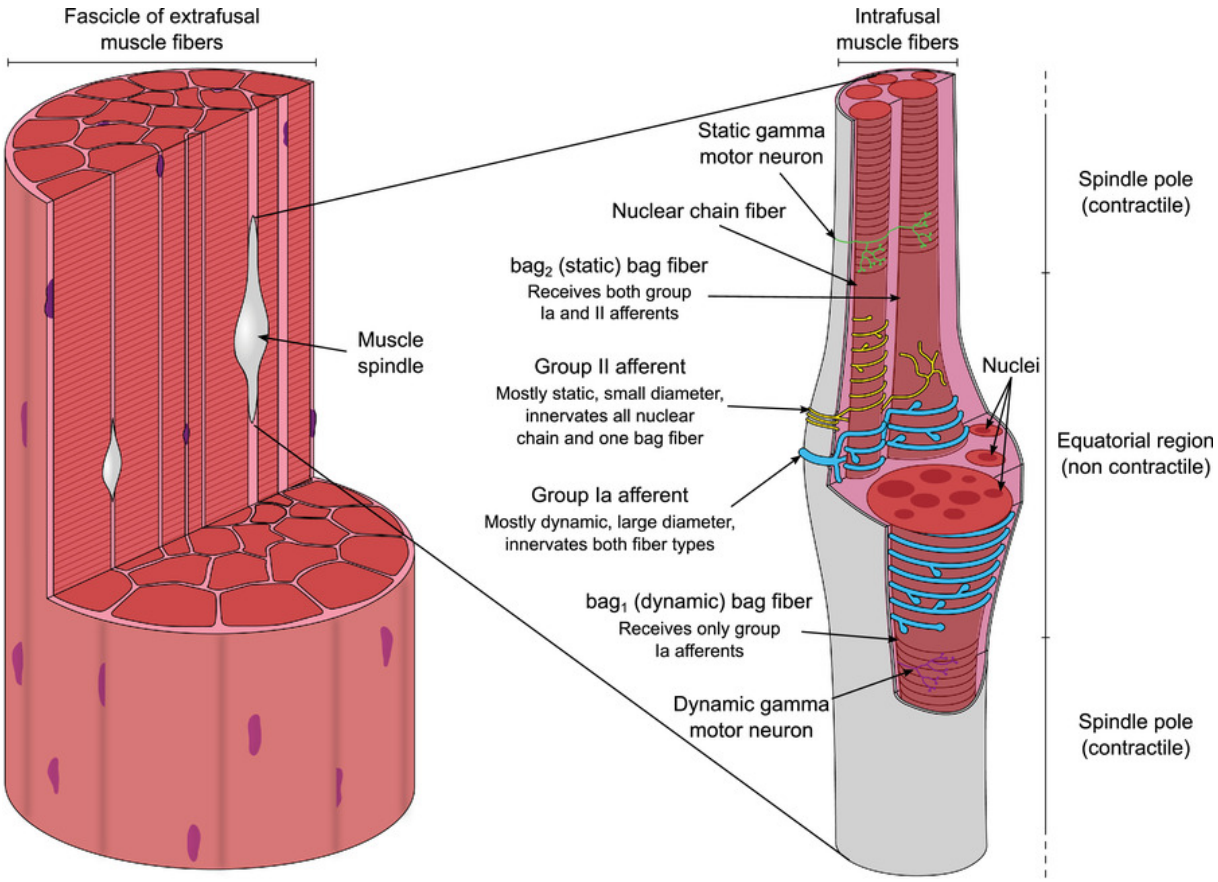

Figure 2.3: The morphology of muscle spindles [19].

When testing reflexes in a clinic or a hospital, they are assessed by the physician observing and evaluating their performance. There is a standard scale proposed by the Neurological Disorders and Stroke (NINDS), where the grading goes from 0 to 4:

- 0: There is no reflex.
- 1: There is a shallow reflex.
- 2: The reflex is within normal range, but on the weaker response side.
- 3: The reflex is within normal range, but on the stronger response side.
- 4: The reflex is too strong.

Areflexia, which means the absence of reflex, although uncommon, usually indicates that there is a lesion in the reflex arc and in motor neuron injuries. The most common cause of areflexia is neuropathy.

Hypoactive deep tendon reflexes are found when there are lower motor neuron injuries. The most common causes of hyporeflexia are hypothyroidism and cerebellar dysfunction.

Hyperactive deep tendon reflexes are a result of upper motor neuron injury, including brain, brainstem, or spinal cord. It usually indicates the existence of damage in descending pathways related to the reflex arc.

Although this is an accepted and standardized practice, clinical deep tendon evaluation is quite subjective to the evaluator and qualitative, which leads to inevitable disparities in its evaluation. However, it is still useful as a primary and simple procedure before performing other necessary tests [20].

2.2 Electromyography

Electromyography is a medical procedure used to measure electrical activity coordinated by the nervous system and produced by the muscles when they contract. It is used clinically to assess the well-being of the nervous system.

The EMG signal is a complex biological signal that varies regarding the anatomical and physiological features of the muscles, the nervous system, and the instrumentation used for its collection.

This signal represents the current that flows through the membrane of the muscular fibers, which, grouped with an α-motoneuron, form the motor units (MU). MU are the functional units of the muscle, and when activated, they will produce the MU Action Potential (MUAP). The added MUAPs are what will cause the muscle to contract and will be recorded and represented by the EMG [21].

The EMG signal is going to change according to the distribution and arrangement of MUs and muscle fibers. These configurations change greatly depending on the muscle and the individual. Muscles may have between 100 and 500 MUs, and inside each MU, the number of muscle fibers can range from around 10 to more than 2000. Spatial configuration is another factor to consider since one motor unit can be between 2 and 10 mm. Therefore, these factors could alter the signal, especially if the EMG is performed with needle electrodes [22].

Motor unit action potentials

The recordings of the EMG represent the electrical field caused by the MUAPs of the sar-

colemma, which is the cytoplasmic membrane of the muscle fibers.

For the muscle to contract, the nervous system has to send a message to the muscle. That signal then reaches the neuromuscular junction, which is the connection between the nervous system and the skeletal muscle. In the neuromuscular junction, we find the axon terminal, almost touching the muscle fiber but separated by a thin space called the synaptic cleft. The axon contains synaptic vesicles which contain a neurotransmitter called acetylcholine (ACh). When a signal reaches the axon, the ACh will be released into the synaptic cleft and will bin to the nicotinic ACh receptors located in the muscle fiber.

Once ACh binds to its receptor, a depolarization occurs as the nicotinic receptors are permeable to sodium and potassium, being the sodium channel conductance higher than potassium.

After the action potential has been propagated and the membrane potential reaches its peak, repolarization begins. Sodium cations channels close and potassium channels open. Now that the outside of the cell has negative charge, the potassium cations will exit the cell following gradient, restoring negative charge excessively within the cell, in order to avoid another action potential right away. During hyperpolarization the cell will reach the resting potential.

The propagation of those action potentials through the T-tubules to the sarcoplasmic reticulum, where calcium levels rise. The calcium binds to the troponin molecule, which will cause muscle to contract. [23] .

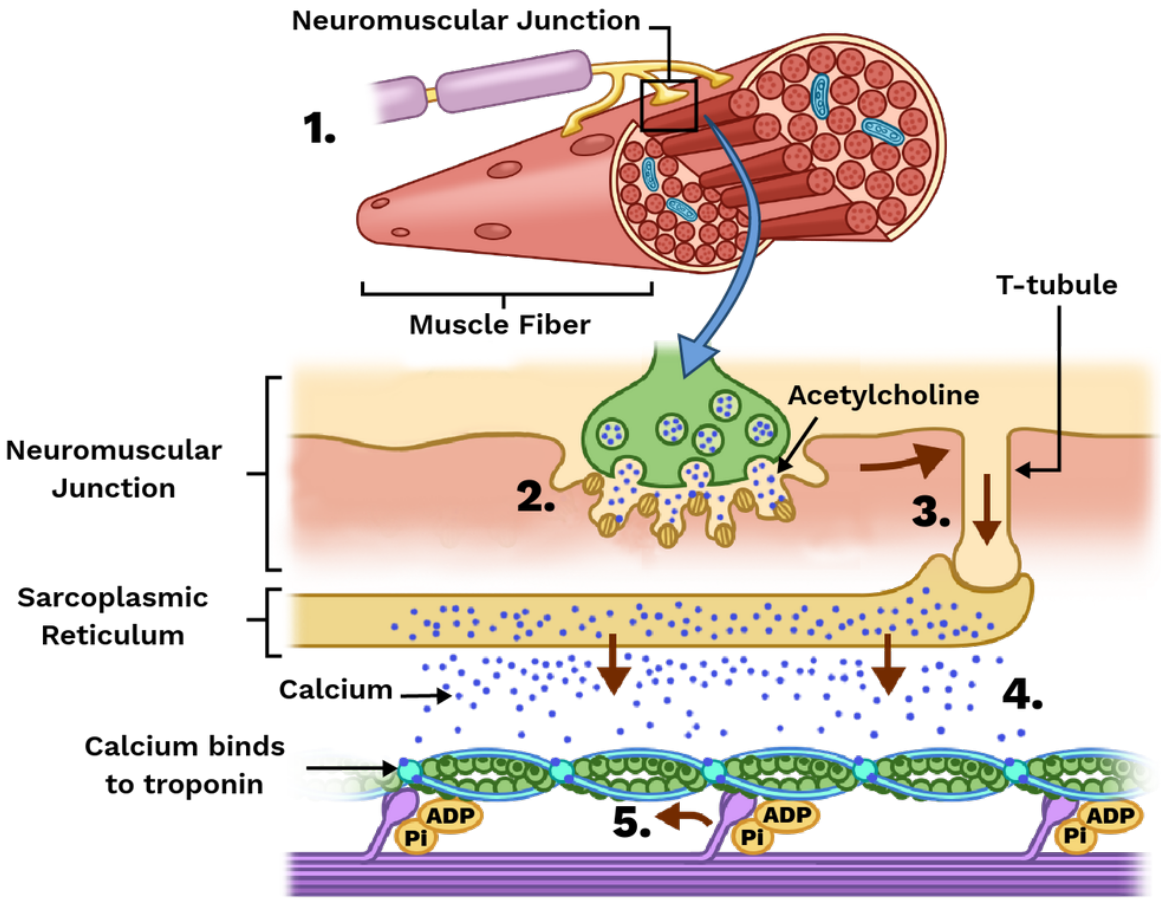

Figure 2.4: Propagation of an action potential through the neuromuscular junction [24].

Types of EMG

1. Surface EMG: This type of EMG is obtained using non-invasive electrodes, which are placed directly on the skin, without any opening or insertion needed. There are three electrodes placed on the skin, one of them serves as the reference electrode for the other two.

The size of these electrodes may vary among brands, but they are usually between 1 and 5 centimeters in diameter, although a wider range of sizes are available depending on the purpose. The decision regarding the size is based on the muscle we want to study (the smaller the muscle, the smaller the electrodes), the features of the muscle, or the distance between electrodes we may want for each evaluation. Another factor to consider when choosing the size is that the larger the electrode, the more information is lost, as the action potential recorded is the mean of the measurements of each electrode.

Regarding materials, electrodes are made out of silver or silver chloride combined with an electrolyte gel solution which improves the impedance reduction and ensures a more stable connection between the skin and the electrode. This stability allows for a higher amplitude of the signal, improving the noise to signal ratio.

Concerning placement, there are two options: monopolar or bipolar placement. The signal detected from monopolar electrodes is extracted from the electrical potential of the skin above the muscle, compared to the electrical potential above bone. This arrangement is susceptible to crosstalk, such as interferences from other muscles, but it provides real potential. On the contrary, bipolar electrodes detect the signal from the difference between two spots on the muscle, which reduces crosstalk but is not as sensitive as monopolar placement [25].

2. High density EMG (HD-sEMG): This is a type of EMG that, unlike surface EMG, uses a large number of electrodes placed close to one another directly above the muscle. The abundance of electrodes enables HD-sEMG to capture electrical signals from various segments within the muscle, offering a more detailed insight into muscle activation compared to conventional surface EMG. One of the advantages of this kind of testing is the possibility of not only assessing the electrical activity of the muscle but also obtaining information about the spatial distribution of that electrical activity. This meticulous analysis is facilitated by the dense arrangement of electrodes [26].

3. Needle EMG: This type of EMG is performed with needle electrodes, which are invasive and are introduced inside the body. This procedure follows the same electrode placement as conventional surface EMG. There are different possible options for these electrodes, and each of them has its own characteristics and uses.

The monopolar electrodes record longer and higher amplitude MUAPs; they are cheaper than concentric needles and are less painful for patients. They also allow for fewer disturbances of the signal. The concentric electrodes generate a signal with less noise thanks to the closer distance between the needles, but they can also lose some information in the process.

Within the needle electrodes, the monopolar and concentric ones record only one MUs, macroelectrodes allow to record more MU and more fibers. The advantage that it grants is the opportunity to check a larger area of the muscle. Even so, macroelectrodes are not usually used, and monopolar and concentric needles are the norm when a needle EMG is performed.

Superficial EMG has multiple advantages such as the lack of preparation needed from the

patient, it is not invasive, and it is an easy test, so the medical staff performing it does not need as much preparation. It is a test usually performed in the evaluation of movement disorders or as a first approach when screening muscle weakness or fatigue.

HD-sEMG, although it is not invasive, can be time-consuming, and due to the large number of electrodes, it can also be quite expensive. Nonetheless, it may be useful as an alternative test if more information about the muscle or nervous system is required but needle EMG is not an option.

Conversely, needle EMG is more specific and gives a more trustworthy signal, thanks to fewer interferences. It allows for deeper, sometimes inaccessible information of the signal compared to surface EMG, and they are more commonly used in clinical EMG. The downside of this test is that it requires highly trained specialists, it inflicts more pain, and it is more invasive on the patient [22].

EMG applications

EMG is mainly a clinical diagnostic tool for assessing nerve or muscle malfunction, such as muscular dystrophy or nerve compression syndrome. It allows for the identification of abnormalities in the electrical communication. In the recent years, EMG has also been used in the prosthetics and assistive technologies field. EMG signals can be used to control prostheses, robotic exoskeletons, or other rehabilitation devices. Even Human Computer Interface, which could enable users to control interfaces through muscle movement, is being researched [27].

2.3 Reflex hammers

The use of percussion on the body has been employed for centuries to determine free fluids inside the body. Initially, these percussions were performed by tapping the fingers onto the surface the doctors wanted to examine. Years later, the first devices to perform percussions were created, from the pleximeter, which was placed on the chest and patted with the fingers, to the first percussion hammers.

In the mid-19th century, the percussion hammer, Wintrich hammer, was introduced. Later on, it underwent updates in its weight, shape, and size until the early 20th century when the first hammer specifically designed for tapping tendons, called the 'reflex and sensibility tester,' was introduced.

As reflex testing became an usual procedure in hospitals, more interest was placed on the

development and improvements of reflex hammers. A multitude of materials were tested for both the handles, such as wood, whalebone, and different metals, and the head, like wood, rubber, or lead. Other modifications were also implemented, with the most remarkable being the weight because, at first, the hammers were too light and resulted in unsteady results.

Even though there have been improvements in materials and ergonomics, multiple reflex hammers are used clinically nowadays, all of which were introduced in the late 19th century and the beginning of the 20th century:

- **The Taylor Hammer:** Introduced in 1888, was the first real reflex hammer ever created. It had a rubber head in the shape of a triangle, with the wider side for larger tendons and the narrower side for smaller tendons. It also incorporated a pointed end in the metal handle to check for cutaneous reflexes.

Figure 2.5: Picture of a Taylor reflex hammer.

- **The Babinski hammer:** Was initiated in practice after 1896, with two versions. Both had the same nickel-plated steel handle but differed in the head arrangement. The first one had a circular rubber-protected head perpendicular to the handle, and the other had a rect-

angular plate shape, oriented in the same axis as the handle, which was very convenient for transportation.

Figure 2.6: Picture of a Babinski reflex hammer.

- **The Troemner hammer:** Presented in 1910, was metallic with interchangeable rubbery knobs on both sides of the head. It also had a wider knob designed for larger tendon testing such as Achilles, patellar, or triceps tendon, and a smaller knob for testing tendons such as biceps humeri or femoris. It also included a sharper end of the handle for cutaneous reflex testing.

Figure 2.7: Picture of a Troemner reflex hammer.

- **The Berliner hammer:** Also introduced in 1910, has an axe shape, fabricated in nickel with a rubber protection on the rounded part of the head. The distal part of the handle could also be used for superficial reflex testing.

Figure 2.8: Picture of a Berliner reflex hammer.

- **The Queen Square hammer:** Introduced in 1925, had a bamboo handle with a perpendicular metallic disk with the edges coated with rubber and a pointed handle for cutaneous reflex testing. Because the handle is longer and more flexible, it has a painless tap [12] [28].

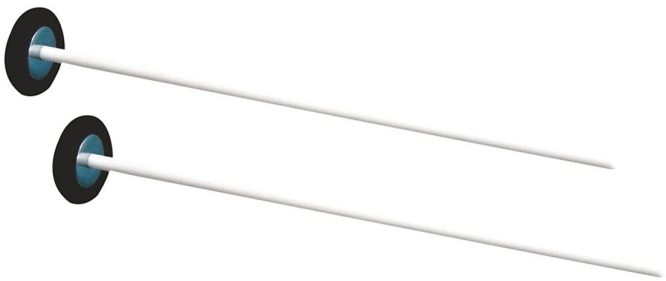

Figure 2.9: Picture of a Queen Square reflex hammer.

Although they are used nowadays, not all are equally useful for every task. In the study [29], the Taylor hammer, the Queen Square hammer, and the Babinski hammer were quantitatively

assessed to find discrepancies among the tap force they deployed. They concluded that due to the smaller weight and handle, the Taylor hammer was the least appropriate to deliver a larger peak tap force, especially if hyporeflexia is suspected. The one that had the best results was the Queen Square hammer, thanks to the flexibility of its handle and higher weight, followed by the Babinski hammer, which is also useful for plantar reflex evaluation (Babinski sign) and detecting neurological abnormalities.

However, because the Taylor reflex hammer has multiple qualities, like its price or its compact shape, it is used for testing knee and elbow reflexes. Conversely, the Troemner hammer does a better job at evaluating smaller tendons, such as fingers and toes. The Berliner hammer is used for its great adaptability in reflex evaluation [30].

Sensor reflex hammer

As discussed earlier, reflex hammers and the current evaluation of reflexes are subjective and qualitative, leading to unwanted variability of results. This is why, even though they are not used clinically and only used for research, reflex hammers with incorporated sensors to measure the force exerted could be useful to standardize the procedure.

There are some options in the market, such as the ADinstrument hammer, the sordalab hammer, or the Vernier hammer. The difference between the first two and the third one is the wireless option. The Vernier hammer has a built-in wireless sensor, while the ADinstrument and the sordalab have a cable in order to connect them to a computer.

Some of the improvements this sensor attachment could generate include the ability to store, analyze, and compare data. It can also be useful for teaching, allowing students to realize how much force they are exerting on the tap, and it provides room for quantitative investigation. If matched with an EMG, a deeper study of the patient's response can be evaluated. How the patient responds to a stronger or weaker tap could be tested, and it could be the basis for a more precise diagnosis.

Despite the clear advantages, sensor-incorporated reflex hammers do not fix all the inconsistencies regarding tendon tapping. For example, they do not address the difference in angle orientation, or the torque exerted by the physician, nor the accuracy of the aim of the tap towards the tendon.

They also have other disadvantages regarding practicality in the clinic. For example, because they need cables for the sensors, they are more heavy and uncomfortable to carry. Some of them need to be connected to a computer, so there is need for an electrical current and prox-

imity to a computer. There is also a time downside compared to the traditional procedure, as a lot of time would be lost due to program opening, software checks, and if accompanied by the EMG, electrode placing, among others.

Nevertheless, plenty of medical procedures take time and require electrical current, such as an electrocardiogram, or a Magnetic Resonance Imaging, which are worth the effort and time because the information they provide is highly diagnostically valuable. Sensor reflex hammers as a complement of an EMG could start being used if the advantages and information that they provide exceed the disadvantages.

2.4 Current approach

Most of the sensor reflex hammers, including the 'PowerLab15T' available in the market, come with commercial softwares that enable the user to perform certain functions. Additionally, there are free GUIs and other types of software that process EMG signals and allow users to perform calculations such as latency or response time.

Although they do not address the same objectives as those covered in this Final Degree Project, there are similar solutions available for latency calculations in reflexes, involving the development of different softwares:

- **Current solution 1.** H-Reflex measurements, using electrical stimulus and EMG: In the case of this project, the reflex is provoked by an electrical stimulus, and the study of the H-reflex is performed with an EMG. Although it is written in MATLAB, they also developed a GUI that allows the user to automate some tasks performed manually before. One significant difference is that in the case of the H-reflex, both the stimulus and response belong to the same EMG signal, contrary to our project, where the stimulus is recorded by the hammer sensor [31].

- **Current solution 2.** A similar reflex assessment was conducted using 'Labscribe'. In this laboratory, they also calculated reflex latency, focusing on Achilles and patellar reflexes, among other analyses such as conduction velocity and amplitude calculations. They manually performed the calculation, with differences in the onset points. The latency was measured from the beginning of the hammer tap to the onset of the reflex response [32].

- **Current solution 3.** Current lab's approach: at the university's laboratory, they export the signals to a spreadsheet software and perform the calculations manually, using commands

and plotting functions.

- **Current solution 4.** The EMG and sensor reflex hammer come with commercial software called 'LabChart'. This is an acquisition and analysis software that enables multiple functions, such as viewing different channels or graphs simultaneously, or segmenting each hammer stroke and EMG response. The disadvantage of this software with respect to latency calculations is that it is not automated; the user has to manually acquire the time where the peaks of both the reflex hammer signal and the EMG signal happen, and subtract them. The LabChart software is not very intuitive and can be difficult to use without proper training.

2.4.1 Limitations of previous solutions

The current approaches, despite their validity and reliability, have some limitations regarding their range of possibilities, medical validity, or lack of automatization. From the previous solutions explained above, none of them have the same approach as this project. All of them are focused on the calculations of muscle latency, but none of them pursued the automatization for the same type of reflex and stimulus.

- The lack of GUI restricts the involvement and interaction between the user and the program, limiting the use of the program to people with some programming knowledge.
- Using only electrical stimuli to induce reflexes, although useful, limits the study of the patient's reaction to other kinds of stimuli, like mechanical or visual.
- Performing the calculations manually with a not intuitive tool is time consuming and can lead to human error.
- Using commercial software is expensive, and taking into account that it also does not perform latency calculations automatically means that it would not only significantly increase the cost of research, but it would also not solve this specific problem.

Due to all these limitations of previous approaches, this Final End Degree emerges to overcome some of these obstacles and offer solutions to actual necessities. This project approaches the goal of processing the signals and calculating muscle latency after a mechanical stimulus while implementing a GUI for improving the user's experience.

Chapter 3

Methods

In Chapter 3, the methods used to develop this project are described in detail. This Chapter begins with an explanation of how and why the project was initiated, providing context and background information about the initial data and approach.

After introducing the problem, the specific features, functions, and algorithms employed throughout the project's development are examined. The tools used are explained with respect to their purpose and advantages as part of the methods contributing to the fulfillment of this project.

3.1 Problem identification

As discussed in Chapter 1, this project aims to develop a tool for researchers to better calculate the latency of the patellar reflex caused by the hammer stroke to the patellar tendon. This is a time-consuming task if performed manually or using a spreadsheet software, and it is easily automatable. To understand the processes involved in completing the project, an explanation of the type of file used and the information it contains is provided.

Input file structure

The signals were extracted from the EMG and reflex hammer equipment located at the Alcorcón campus of Universidad Rey Juan Carlos (URJC), which are used for various types of physiological investigations. The information obtained from the Powerlab15T [3] device has some features that will be discussed below.

Let's consider first the structure of the files provided by the Powerlab15T device, which contain information about the measurements and the actual measured data. The first lines of the file contain formatting information, which provides details such as the date, time, and other signal features such as the sampling interval which, is 1 millisecond (ms). With a sampling interval of 1ms, even a short duration test, for example one minute, contains thousands of data points.

Additionally, signal range information is provided within the formatting data. The amplitude of the reflex hammer signal ranges between 0 V and 10 V, except for negligible negative values due to external noise. The amplitude of the EMG signal ranges between -2.5 mV and 2.5 mV.

After formatting information, the file content is given by the measured data. This data consists of three columns: the first column containing data time, the second column the reflex hammer signal, and the third column the EMG signal.

Software functions

After the analysis of the file format and the structure of the measured data, we developed the processes described in the following, in order to achieve the objectives described in Chapter 1.

First, a visual tool is required for the user to interact with the signals and other useful functions. With this goal in mind, an interactive GUI is intended to enhance the user experience, allowing different functions to be carried out. The GUI should enable the user to choose the file they want to use and import the information into the program.

After the file is selected, both signals should be clearly represented so that the user can benefit from a visual representation. To improve user interaction with the signal representations, options should be added, including which traces to show and hide and which filter to apply to the EMG. For this purpose, the program should have different filtering options for the EMG, so the user can choose and compare different signal processing options. Another feature desired for the program is a toolbar with options to amplify and navigate through the graph, allowing the user to select the signal segment where the latency will be calculated.

The most important function that the GUI should provide is the latency calculation, an automated and reliable task for the segment and processing method required by the user. Finally, to facilitate its use and installation, the program should be provided in an executable format for Microsoft Windows.

3.2 Software development

3.2.1 Python and Python libraries

Python is a powerful and versatile programming language that includes multiple libraries and modules to help the user develop a multitude of tasks and programs. One of the most important advantages of Python, and the reason why it was used in this project, is that unlike other languages, such as MATLAB, Python is an open-source language.

Another advantage of Python is its portability. Python is independent of any operating systems and can be operated on different platforms or code editors, making it more appropriate for people in different departments or institutions to interact with the code. The only aspect to be aware of, is that the libraries used in the program will have to be installed on each computer.

Also, this project is open to researchers' needs, meaning that they should be able to modify or add features to the program as the investigation or task requires. For these reasons, Python is the appropriate choice, because it is a simple language, widely used in the healthcare field. In order to develop the program, some python libraries were implemented.

Tkinter Library

This library is a tool for creating GUIs in Python. One of its main features is the possibility to create widgets like buttons, menus, checkboxes, message windows, and labels, among others. All these widgets are customizable to the programmer's preferences and allow coupling functions to different kinds of buttons, making the program interactive and personalized.

Tkinter is also compatible with other Python libraries, which allows a good combination between the GUI's functionality and the developer's needs, like data processing and representation. Also, this library is implementable on Mac, Linux, and Windows, which is a fundamental characteristic for an open-access program.

This library was used to create the GUI for this project. Thanks to the multiple tools it provides, a complete interface with plenty of useful features was developed.

Matplotlib Library

The *Matplotlib* library specializes in high-quality graphical representation of functions and signals. The visualizations that this library offers may be static, animated, or interactive, making

it a useful tool for data visualization.

It is highly customizable and has multitude of plotting possibilities, such as line plots, dot plots, or histogram plots, among others. It is also easy to integrate with the *NumPy* and *SciPy* libraries, explained in the following sections. Furthermore, it was created to work easily with electroencephalography (EEG) data, so it had a healthcare objective from the beginning.

This library allowed for the plots of the signals to be represented, supplying the tools to name the axes, title the graph, create a legend, plot the graphs on different axes, color the graphs, or decide the shape, width, and transparency of the plots, among others.

NumPy Library

NumPy is a numerical computation Python library specialized in numerical calculus and data analysis, known for its flexibility and speed. It is particularly useful when working with arrays. *NumPy* is also useful for manipulating arrays, generating random numbers, vectorization, indexing, and performing operations between different size arrays.

This library is useful for for setting index ranges, assigning data to arrays, and calculating, for example, the mean of the array.

SciPy Library

This library is used for scientific and mathematical computations. It is built on *NumPy*, amplifying its capabilities when it comes to data reconfiguration and processing. It is a useful tool for performing integration, optimization, linear algebra operations, or statistical operations. The most important features it provided for this project are its built-in signal processing functions, especially regarding filters and finding the peaks of the signals.

Tabulate Library

Tabulate is a Python library that enables the creation of formatted tables from different sources, such as *NumPy* arrays, lists, or tuples. It has the advantage of providing a simple language to create tables and offers personalization and customization options. This library was used to design the table that the program returns once the latencies are calculated, in order to display it.

PyInstaller Library

This library is used to create an executable file from the python code. An executable will

allow the user to use the program and all the features it offers without the need to install a code editor, protecting the code and making the program more accessible [33].

3.2.2 Graphical User Interface (GUI)

A GUI is a digital interface designed to be interactive for users, thanks to visual elements such as menus, message boxes, or windows. These interfaces serve as the intermediary between users and computers, allowing users with basic knowledge of informatics to analyze and work with data in their field, as physiology and medicine in the case of this project.

Thanks to their simple graphical elements and interactivity, achieved through the use of icons and buttons, GUIs are used by most operating systems and software applications, as for example Microsoft Windows, Linux, and Apple's macOS.

GUIs work following the model-view-controller principles [34], which have one main characteristic: they do not show the same information to the user as the actual information used by the hardware. Therefore, users can only choose between the options presented to them by the GUI.

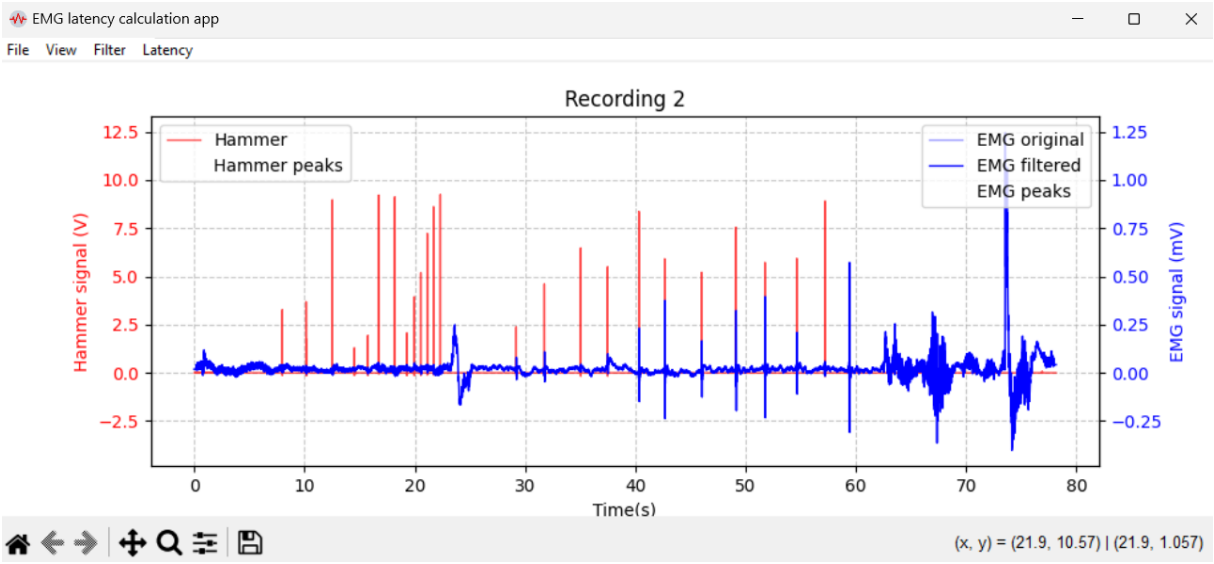

Figure 3.1: The Graphical User Interface. EMG (blue trace) and hammer (red trace) signals are plotted as an example. GUI elements, e.g. buttons, menu and graph, are described in the text.

The elements provided by the GUI developed in this project can be appreciated in Figure

3.1, where an example of the visual representation of the reflex hammer and EMG signals is given. These elements are:

Menu

The menu of a GUI is a graphical control element that enumerates the commands the user must select to execute a function. Normally, the options of a menu will have a cascade display, allowing the user to perform some functions.

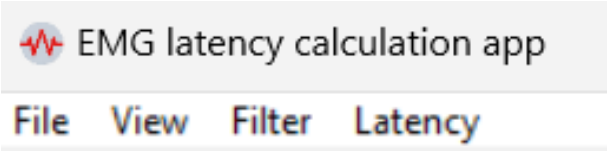

Figure 3.2: Main menu.

The menu bar consists of four tabs, as shown in Figure 3.2. The tabs are:

- **File:** The *File* tab opens a drop-down list with two selectable options: *Open* and *Close*. By pressing *Open* a window browser opens for importing the selected file. By pressing *Close*, the application ends.
- **View:** The *View* tab opens a drop-down list with different selectable options. Every list item has two options, checked and unchecked, whether the user wants to see the trace or not, respectively. In this project they are used as part of the menu to allow the user to decide which signal to plot. The last button, *Reset zoom*, will allow the user to go back to the original signal view.

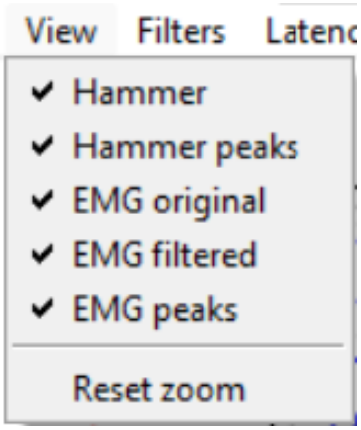

Figure 3.3: Drop-down list of selectable items from the *View* tab.

- **Filter:** This is a single-option menu, meaning that among all the possible options, the user will only be able to choose one. This element is used to enable the user to try different filters for the EMG signal and choose the one that best fits their needs. The trace that will change is the 'EMG filtered', allowing the user to compare the filtered to the original EMG signal. The possible options this menu allows are shown in Figure 3.4.

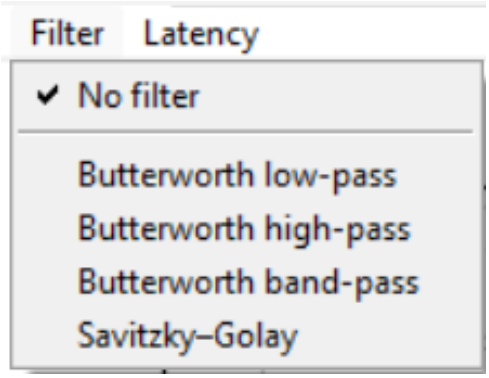

Figure 3.4: Drop-down list of selectable items from the *Filter* tab.

- **Latency:** This menu enables the user to calculate the latency. The user can select the desired definition of latency. The latency can be defined as the time difference between the local maximum of the hammer signal and:
  - The local maximum of the EMG signal.
  - The local minimum of the EMG signal.

- The larger absolute value of the EMG signal.
- The local maximum of the derivative of the EMG.

The possible options enabled by this menu are shown in Figure 3.5. These calculations are explained in Section 3.3.1.

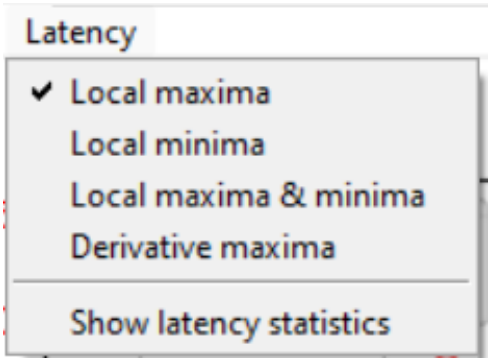

Figure 3.5: Drop-down list of selectable items from the *Latency* tab.

Graph Toolbar

A toolbar consists of a set of icons that represent common functions that the user wants to perform, the toolbar is added to the frame of the graph. The Graph Toolbar shown in Figure 3.6, allows performing the following function (from left to right):

- Reset to the original view.
- Back to the previous view.
- Forward to the next view. These three options only undo the toolbar's zoom and navigation features, if the user wants to go back to the original view after rolling the mouse wheel, they will have to press the *Reset zoom* option located in the *View* menu.
- Navigation tool. If the left button of the mouse is pressed, it will allow the user to zoom in and out. If the right button is pressed, then it will allow the user to move the graph in any direction. If x or y keys are pressed, the respective axis will be fixed, and if the control key is pressed the aspect will be fixed.

- Zoom to rectangle. If x or y keys are pressed, the respective axis will be fixed. This possibility that the toolbar offers, will enable the user to zoom in throughout the graph.
- Configure subplots. This is a default toolbar function that allows the user to move the figure horizontally and vertically within the GUI's window, when the program is implemented in executable form.
- Save the figure. When this button is pressed, the graph will be saved in *.png* format, where the user decides within the file explorer.

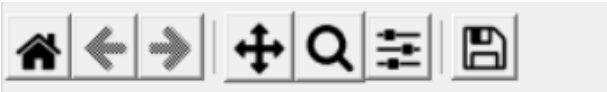

Figure 3.6: Graph toolbar.

3.2.3 Implemented functions and classes

Functions

Specific functions have been implemented into the GUI code, these are described in the following:

- **Select file:** The first step for the user to visualize the signals is selecting the desired file. Therefore, a function will open the computer's file explorer once the program is executed, so the user can select the desired *.txt* file.
- **Read data:** After the file is opened by the user, another function will process the data, performing an initial data cleaning. This function will strip the data from the formatting information. It will also replace ',' with '.' to ensure Python correctly identifies float numbers and decimal separators. The data will then be divided into three arrays: time, hammer, and EMG.
- **Processing signals:** In order to process the signals, a group of functions were implemented. The functions developed in order to process the reflex hammer and EMG signals are explained in Sections 3.3.1 and 3.3.2 respectively.

- **Plotting function:** This function generates plots with processed and original signals, add the x and y labels, and title the graph. The color of the plotted signals will be decided, and a legend will be set. Additionally, the toolbar will be added to the frame, allowing further functions for the user.

- **Zoom function:** This function allows the user to zoom in and out of the graph by rolling the mouse wheel. If only the wheel is being rolled, the zoom is fixed for the y-axis, and the zoom is centered on the mouse cursor, if the wheel is rolled upwards, the graph will be amplified, and if the wheel is rolled backwards, the graph will diminish. If the uppercase key is pressed while the wheel is being rolled, the zoom is fixed for the x-axis. If the 'Ctrl' key is pressed while rolling the wheel, none of the axes will be fixed, so the graph will increase and decrease proportionally. If the 'Alt' key is pressed while rolling the wheel, the zoom is static, but the graph will go to the right when rolled downwards, and to the left when rolled upwards. This feature is accompanied by a reset zoom function located bellow the view options, so the zoom performed with the mouse wheel can be undone, and returns to the original view.

- **Latency function:** This is the core of the program, as it allows for the performance of latency calculations. To achieve this task, a set of functions was developed for calculating the latencies between the reflex hammer stroke and the reflex response. The sub-routines of this function are explained in section Section 3.3.3.

- **Save function:** This function enables the user to save the results of the peaks of the reflex hammer, the time where they happen, the corresponding delays and the mean delay, as a *.txt* file.

- **Create functions:** These functions create the different features of the program. These include the *File*, *Filter*, *View*, and *Latency* tabs, also the canvas, graph toolbar and the main window.

- **Update functions:** In order for the multiple or singular option menus to work, some functions were developed. These functions are responsible for showing and hiding the plotting traces, or for changing the process method of the EMG filtered signal.

- **Close function:** This function enables the user to close the GUI, this option is located below *Open* in the *File* menu option.

Classes

In order for the code to be clearer and more organized, some classes were developed, where these functions are categorized. These classes' names start with the acronym of Latency Calculator Application (LCA), and they are:

- **LCAData:** This class is going to enclose all the data variables, like the filtered signals or the peak values. It also includes the functions that are going to apply the filters and the postprocessing methods to those data variables, and save them.
- **LCAFileUtils:** This class is going to include the functions that allow the user to select the file, read it, save the latency calculation file or show the logo.
- **LCAPlot:** This class includes all the functions and variables related to the plotting features the program provides.
- **LCAPlotWindow:** This class contains all the functions that create the different menus, scrolling and refreshing options, windows, or toolbars among others. Thanks to this class multiple files can be opened at the same time.
- **LCASignalUtils:** This class incorporates all the EMG filters that the program uses, as well as the functions that find and arrange the signal's peaks.
- **LCAStatsWindow:** This class is related to all the GUI's features. It contains all the functions that create the different menus, scrolling and refreshing options, windows, or toolbars among others.

3.3 Signal processing

3.3.1 Reflex hammer signal processing

In order to graph the signals more clearly, a *Matplotlib* tool called *twinx* was implemented, which allows representing graphs on two or more different axes. Figure 3.7, shows how both signals are represented in different axis, with different colors for a more clear visualization. The fact that in their respective units, the hammer's amplitude is around 10 times bigger than the EMG's amplitude was convenient for visualization.

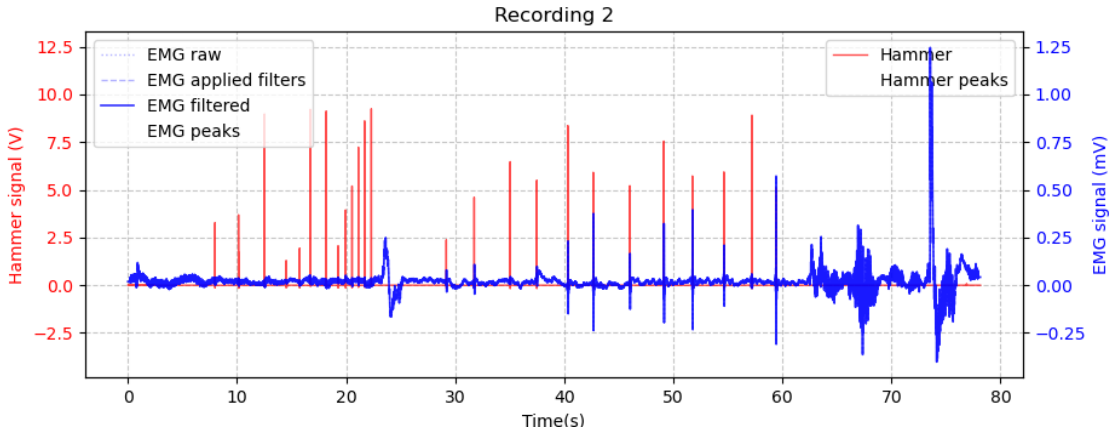

Figure 3.7: Reflex hammer and EMG signal representation in separated axes.

3.3.2 EMG signal processing

In this Section, we describe the functions used to process data after it is imported by the software. The possible types of noise an EMG signal may contain are:

- Unavoidable electrical device noise: This type of noise can only be eliminated using high-quality electronic equipment.
- Electromagnetic radiation: This kind of noise is due to electric and magnetic radiation affecting our bodies.
- Motion artifact noise: Such noise can be caused by an electrode interface or cable. Wisely setting up the device's circuit can help reduce this kind of noise.
- Instability of the EMG signal: The amplitude of the EMG signal is random and depends on the number of active MUs and their firing rate during contraction, which causes noise [35].

After choosing good quality equipment and an effective circuit setup, numerical filtering is another way to reduce the noise-to-signal ratio. Depending on the frequencies where the noise and signal lie and the objectives we want to accomplish, different filters may be convenient or not.

Available filters are: a low-pass filter, a high-pass filter, a band-pass filter and the Savitzky-Golay filter. The first three filters were implemented as Butterworth filters. Python facilitates

the use of these filters through the *SciPy* library and a built-in function called *butter*. This filter has been applied using another *SciPy* function called *filtfilt*, to ensure a zero-phase shift of the signal. The parameters introduced are:

- The order of the filter, which we set to 2.
- The sampling frequency, which is set to 1000 Hz.
- A boolean parameter, which will be True for an analog filter and False for a digital filter. In this case, because our signal is digital, we have set the filter to digital as well.

The cutoff frequency for each of the filters designed are:

- Low-pass filter: 60Hz.
- High-pass filter: 30Hz.
- Band-pass filter: From 30 to 60 Hz.

Low-pass filter

Low-pass filters allow low frequencies to pass while attenuating high frequencies above the cutoff frequency. We have set the cutoff frequency to 60 Hz, indicating that only frequencies below that point will pass after the filter is applied. Low-pass filters will eliminate high-frequency interference noise from external or internal sources, such as high-frequency ambient noise or high-frequency aliasing noise. If the signal values are predominantly in the higher frequencies, information may be lost.

High-pass filter

High-pass filters are characterized by allowing higher frequencies above the cutoff frequency to pass while minimizing frequencies lower than the cutoff. In this case, the cutoff frequency is set at 30 Hz, meaning that frequencies below that value will be filtered out. High-pass filters are useful for eliminating low-frequency noise such as low-frequency interference, electromagnetic noise, artifacts, or low-frequency electrical interference. If the signal contains many low-frequency values, this filter will attenuate and distort the signal.

Band-pass filter

Band-pass filters are a combination of both low-pass and high-pass filters, allowing frequencies to pass only within a specified range. The range of frequencies allowed to pass through will be between 30 and 60 Hz.

Band-pass filters are useful when there are both low and high-frequency noises, and the main part of the signal falls within the cutoff signal range.

3.3.2.1 Savitzky–Golay filter

The next filter available is called the Savitzky–Golay filter, which acts as a low-pass filter. The Savitzky-Golay filter employs a local least-squares polynomial fitting approach to achieve data smoothing. It performs better at maintaining the signal's structure than other smoothing methods such as adjacent averaging.

The Savitzky-Golay filter operates by fitting a polynomial to a subset of data points. For each point in the data set, it takes a window of neighboring points (a small segment of the data) and fits a polynomial of a specified degree to these points. The polynomial is fitted using the method of least squares, which minimizes the sum of the squared differences between the actual data points and the values predicted by the polynomial [36].

To apply this filter, we utilized an option from the *SciPy* library called *savgol_filter*. For this method, we needed to specify a window length, which should be large enough to capture the noise yet small enough to preserve the signal. We chose a window length of 11 and a polynomial order of 3.

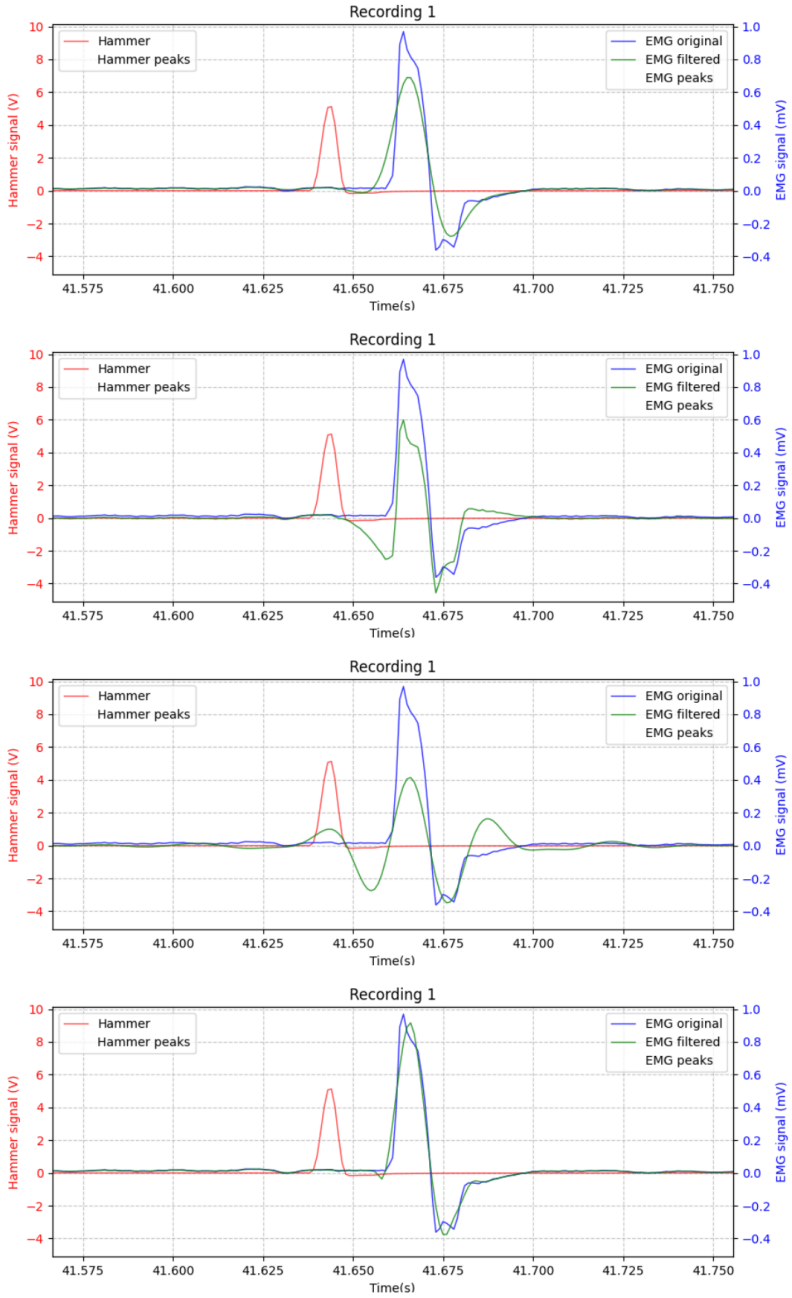

Figure 3.8: Example of the EMG signal after filtering. From top to bottom: low-pass, high-pass, band-pass and Savitzky-Golay.

Figure 3.8, shows a comparison among all the filters available. We can draw some conclusions about them:

- Low Pass Filter: Eliminates high-frequency noise, making the signal smoother and less noisy.
- High Pass Filter: It helps eliminate the baseline and unwanted low-frequency components. Maintains the EMG response's shape, attenuating the amplitude.
- Band Pass Filter: Removes both high-frequency noise and slow low-frequency components, although it distorts the wave.
- Savitzky-Golay Filter: Smooths the signal while preserving important features, such as the peaks and edges.

.

3.3.3 Latency calculation

To correctly perform the calculation of the latencies and accomplish one of the main objectives of this End of Degree Project, a series of functions, processes, and algorithms were implemented.

The function used to detect the local maxima for the latency calculations, is *find_peaks*. This function detects signal's maxima by comparison of neighboring values. To avoid unwanted noise interfering with the algorithm, a threshold was set, below which, no maxima is detected by the function.

To prevent unwanted subject movements from being confused with the actual responses, a time range of 0.5 s is established for all methods. Therefore, only the EMG activity within the time range will be included when calculating the latency. The four methods available for the user to calculate latencies are:

Local maxima

To perform these calculations, the maximum values of the reflex hammer and EMG signals, as well as the times at which these occur, are calculated. After this data is obtained, the latency is computed by subtracting the hammer's peak time from the EMG's peak time: Latency = EMG's peak time - Hammer's peak time.

Local minima

The objective of this method is to find the signal's minimum in order to compute the latency from the maximum of the reflex hammer to the minimum of the EMG signal. To identify the minimum of this signal, the *find_peaks* function was employed, utilizing the EMG signal multiplied by -1 as the input. Subsequently, the latency is determined by subtracting the peak time of the hammer from the moment when the minimum of the EMG signal occurs.

Local maxima and minima

This method is going to calculate the latency between the reflex hammer's maximum and the largest absolute value of the EMG signal.

To identify the maximum and the minimum, the *find_peaks* function will be implemented again, but this time the input will be the absolute value of the EMG signal. This way, both relative extrema will be identified, and the latency will be calculated by subtracting the time at which the largest absolute value occurs minus the time at which the hammer's maximum occurs.

As shown in the third graph of Figure 3.9, both the maxima and minima are identified, and the higher magnitude is the one that will be selected for the calculations.

Derivative maxima

This method is based in [32], where they calculate latencies from the onset of the hammer's signal, to the onset of the EMG's signal.

To facilitate the automation of this task, latency is calculated from the maximum of the hammer to the maximum of the derivative of the EMG, that is, to the point of greatest growth of the signal. Therefore, if there is variability within the same record, another method of calculating latencies could be applied.

To find the derivative of the EMG signal, the gradient function from the *NumPy* library was used. The derivated signal will then be the input to the *find_peaks* function, which will return the EMG's derivative maxima. Once this data is obtained, the latency is calculated by subtracting the time at which the EMG derivative's maximum occurs minus the time at which the hammer's maximum occurs.

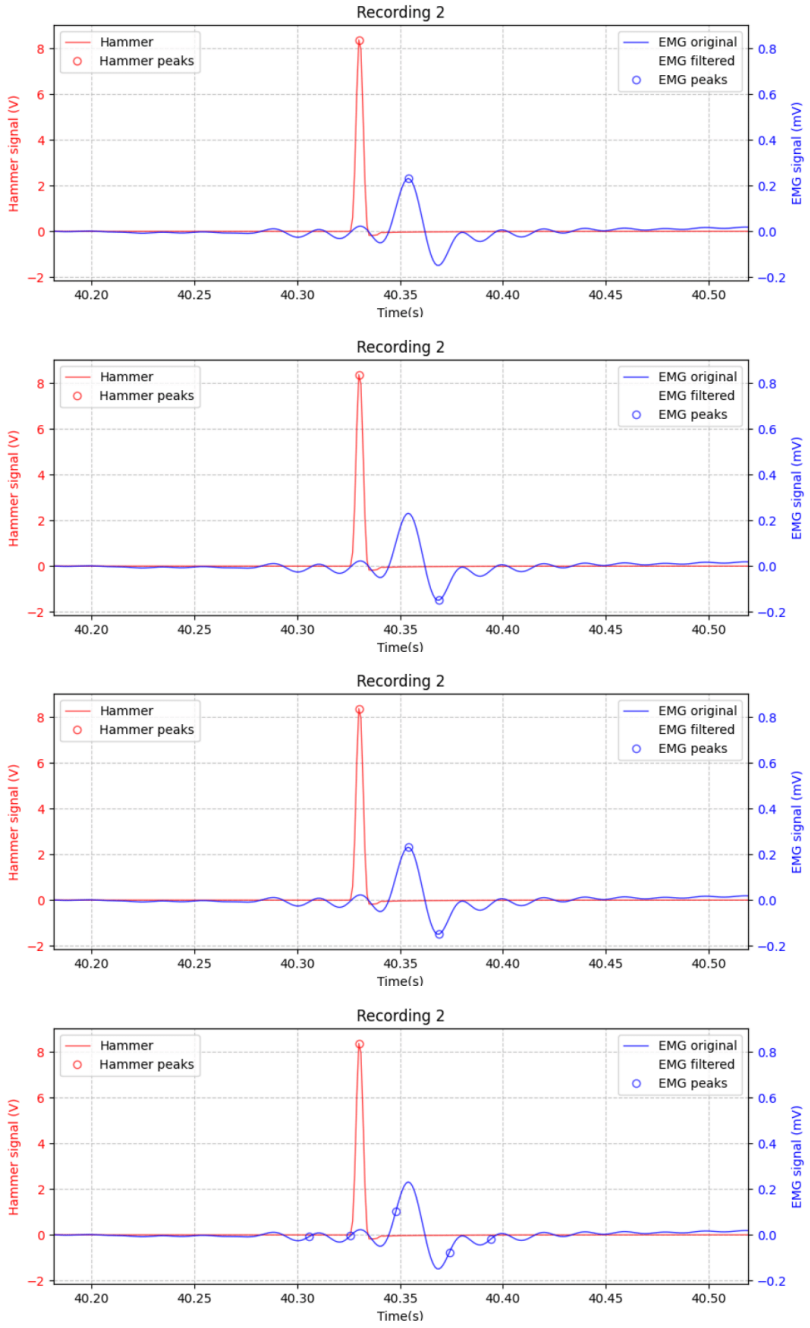

Figure 3.9: Example of the EMG signal with different latency calculations. From top to bottom: local maxima, local minima, local maxima and minima, and derivative maxima.

Depending on the shape of the signal, the desired output or the experiment's requirements, the user will be able to decide which option suits their necessities better, allowing for greater

flexibility and adaptability in the analysis process.

Another important detail to take into account is that if the user wants to process the EMG signal before performing the latency calculations, they will have to select the desired filter beforehand, and the calculations will be performed with the filtered version of the EMG signal.

Latency window's File menu

- Once the user has pressed the *Statistics* button, and the pop-up window is displayed, the user will have the option to save it as a *.txt* file thanks to a *Save as* option located within the *File* menu of the pop up window. After the user presses this option, the program will enable the user to choose the name and location to save the file.
- Below the *Save as* option, the user will have the *Close* function, which closes the calculation window.

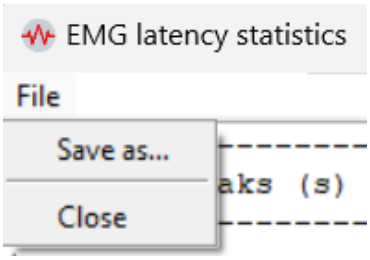

Figure 3.10: Drop-down list of selectable items from the *File* tab of the *Latency* window.

Chapter 4

Results and discussion

4.1 Analysis and interpretation of results

The main objective of this project is to develop a reliable, intuitive, and useful tool for researchers to effectively calculate the latency of the patellar reflex after a reflex hammer tap to the patellar tendon in humans. This task was accomplished by developing a GUI with the programming language Python, and the required libraries. In order to reach this main goal, secondary objectives have been defined and their execution is described in detail below.

Secondary objective 1. Development of a GUI.

The first secondary objective to be carried out was the development of a GUI enabling the user to interact with the signals, features and functions needed to visualize the signals, process them, and calculate their latencies. This GUI includes a menu, with a *File*, *View*, *Filters* and *Latency* options.

Secondary objective 2. Import Data.

This objective consisted in the connection of the program with the previously extracted data. In order to do so, a file-open menu option was enabled so the user could open the desired *.txt* file. This function was complemented by the separation of data and formatting information of the file.

Secondary objective 3. Plot data.

This objective consisted in graphically showing data so the user can have a visual representation of the signals. Once the user has chosen the file, the program will show all the reflex hammer and the EMG signals visualization options, which are:

- The hammer signal.
- The hammer peaks.
- The EMG original signal.
- The EMG filtered signal.
- The EMG peaks (of the processed signal).

The reflex hammer-related features appear in red, as the legend states, and the EMG-related features appear in blue. Also, because the signals have different units and magnitudes, both signals are separated in different axes. Along the left side, in red color, the hammer's axis expressed in Volts is placed. On the right side, in blue color, appears the axis of the EMG signal expressed in millivolts units.

At first, when the file is selected and opened, all the possible features will appear. The user can then hide the unwanted features by deselecting the feature in the *View* menu.

Secondary objective 4. Processing of the signals.

The following objectives to accomplish were the processing and post-processing or latency calculations. Thanks to the *Filter* submenu, the user can experience how these processing techniques modify the signal and delay calculations. These methods and their consequences are explained in detail in section 4.1.1.

Secondary objective 5. Latency calculations.

The calculation of the latencies and method comparison among them are explained in detail in Section 4.1.2.

Secondary objective 6. Executable.

Because the main target users of this program are researchers, another main goal for this project is to develop an executable file. This way the user does not have to install a Python code

editor and can run the program on its own. The executable file comes as a *.exe* file and runs in Microsoft Windows Operative System.

4.1.1 Processing methods comparison

In this Section, we compare how different filters and processing methods affect the EMG traces and the latency calculations.

First, the original signal is analyzed for ten EMG responses of a specific record. The ten recordings chosen can be seen in Figure 4.1, and the latency calculations of the records are presented in the Figure 4.2.

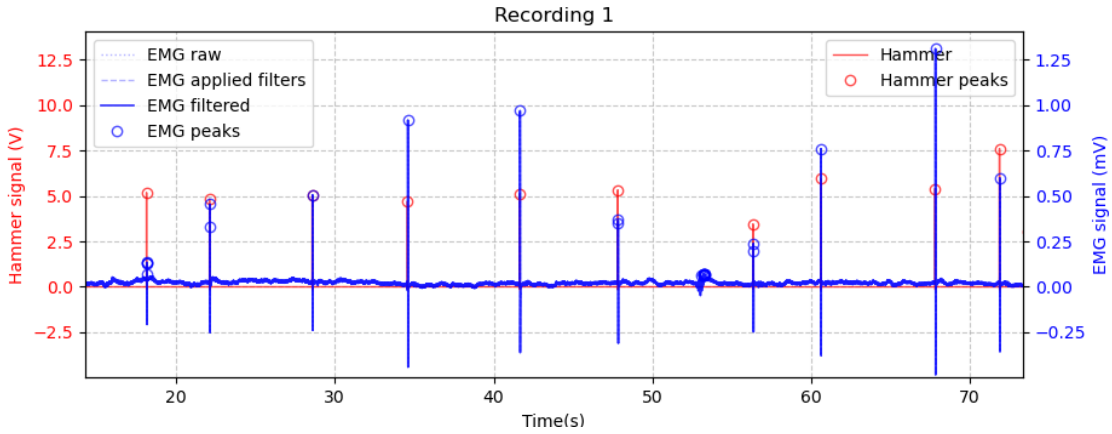

Figure 4.1: Original (unfiltered) hammer (red line) and EMG (blue line) signals considered for comparison of processing methods. A temporal window with ten hammer taps is considered.

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
18.138	5.185	18.161	0.136	23.0
22.104	4.828	22.125	0.454	21.0
28.576	5.015	28.597	0.506	21.0
34.576	4.711	34.598	0.917	22.0
41.644	5.118	41.664	0.969	20.0
47.816	5.328	47.837	0.368	21.0
56.335	3.455	56.356	0.238	21.0
60.592	5.984	60.613	0.759	21.0
67.802	5.405	67.822	1.31	20.0
71.856	7.608	71.877	0.599	21.0

Mean latency: 21.1 ms

Figure 4.2: Statistics corresponding to the signals shown in Figure 4.1.

As shown in Figure 4.2, the mean of the original latency calculations is 21.1 ms, the lowest value is 20.0 ms, and the highest value is 23.0 ms. Both the mean and the individual latency values are around 21.0 ms, which is a value within the normal range for human response time after stimuli [37].

In the following, we present the mean latency values calculated when different filters are applied to the the trace shown in Figure 4.1.

Low-pass filter

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
18.138	5.185	18.163	0.118	25.0
22.104	4.828	22.127	0.328	23.0
28.576	5.015	28.599	0.383	23.0
34.576	4.711	34.599	0.676	23.0
41.644	5.118	41.665	0.689	21.0
47.816	5.328	47.84	0.319	24.0
56.335	3.455	56.358	0.181	23.0
60.592	5.984	60.615	0.548	23.0
67.802	5.405	67.824	0.959	22.0
71.856	7.608	71.879	0.435	23.0

Mean latency: 23.0 ms

Figure 4.3: Statistics of the ten reflex responses after low-pass filtering of the signals shown in Figure 4.1.

High-pass filter

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
18.138	5.185	18.168	0.085	30.0
22.104	4.828	22.125	0.269	21.0
28.576	5.015	28.597	0.29	21.0
34.576	4.711	34.598	0.56	22.0
41.644	5.118	41.664	0.599	20.0
47.816	5.328	47.843	0.209	27.0
56.335	3.455	56.356	0.139	21.0
60.592	5.984	60.613	0.462	21.0
67.802	5.405	67.822	0.795	20.0
71.856	7.608	71.877	0.372	21.0

Mean latency: 22.4 ms

Figure 4.4: Statistics of the ten reflex responses after high-pass filtering of the signals shown in Figure 4.1

Band-pass filter

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
18.138	5.185	18.163	0.072	25.0
22.104	4.828	22.127	0.196	23.0
28.576	5.015	28.599	0.225	23.0
34.576	4.711	34.599	0.412	23.0
41.644	5.118	41.666	0.414	22.0
47.816	5.328	47.84	0.201	24.0
56.335	3.455	56.358	0.117	23.0
60.592	5.984	60.615	0.333	23.0
67.802	5.405	67.824	0.576	22.0
71.856	7.608	71.879	0.266	23.0

Mean latency: 23.1 ms

Figure 4.5: Statistics of the ten reflex responses after a band-pass filtering of the signals shown in Figure 4.1.

Savitzky–Golay filter

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
18.138	5.185	18.166	0.146	28.0
22.104	4.828	22.127	0.398	23.0
28.576	5.015	28.599	0.482	23.0
34.576	4.711	34.599	0.884	23.0
41.644	5.118	41.666	0.916	22.0
47.816	5.328	47.842	0.355	26.0
56.335	3.455	56.358	0.214	23.0
60.592	5.984	60.615	0.689	23.0
67.802	5.405	67.824	1.26	22.0
71.856	7.608	71.879	0.589	23.0

Mean latency: 23.6 ms

Figure 4.6: Statistics of the ten reflex responses after Savitzky-Golay filtering of the signals shown in Figure 4.1

We resume here the calculated mean latencies with different filters:

- Original trace: 21.1 ms
- Low pass filter: 23.0 ms
- High pass filter: 22.4 ms
- Band pass filter: 23.1 ms
- Savitzky-Golay filter: 23.6 ms

Even though filters can be useful for signal visualization, all the applied filters change the mean latency with respect to the original mean latency, when applying the local maxima latency calculation method.

This is because if the filter smooths the wave, like the low-pass, or the Savitzky-Golay filters, in signals where the peak is wide and flattened, the filter will create a maximum that may not correspond with that of the signal.

In filters where the shape of the signal is maintained, as in the case of the high-pass filter, it can happen, that one of the signal's peaks is attenuated more than another, also changing the delay with respect to the original signal.

Therefore, the user must determine their needs and find the filtering and latency calculation method that best adapts to the variability of their recordings, the inconsistencies of the curves, and their objectives.

4.1.2 Latency calculation methods comparison

Because this program enables the user to choose among different methods to perform the latency calculations, we can analyze how this different methods affect the latency values.

Local maxima

Because some of the EMG signals have differences among recordings, two EMG responses will be analyzed to see how the different methods change the calculation. The first EMG response will be smoother, with one maximum per response (Figure 4.7), and the second EMG response will have some variability (Figure 4.9). The signals analyzed will be original, without any processing techniques applied.

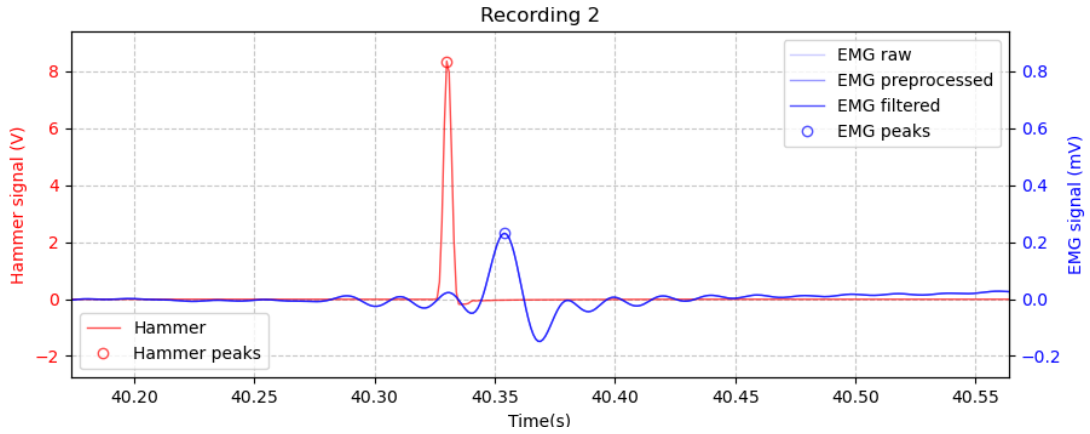

Figure 4.7: Visual representation of the local maxima method of latency calculation for smooth EMG response.

As shown in Figure 4.8, the latency calculations of Figure 4.7, the algorithm correctly detects the maxima of the EMG response and calculates the latency, without difficulty.

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
40.33	8.356	40.354	0.231	24.0

Mean latency: 24.0 ms

Figure 4.8: Latency calculation using the local maxima method of the EMG response shown in Figure 4.7.

On the contrary, the second EMG response, has a less smoother wave, as shown in Figure 4.9. Three maxima are identified by the program, which can vary the correct calculation of the latency.

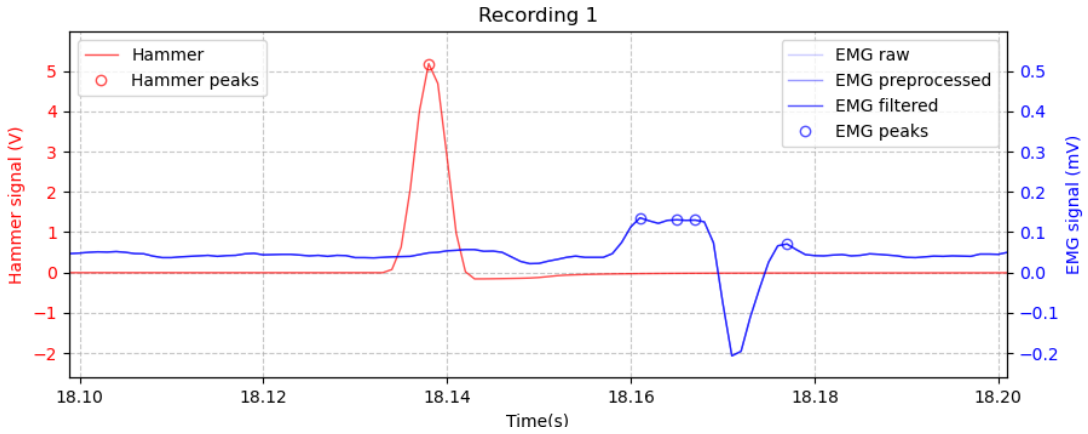

Figure 4.9: Visual representation of the local maxima method of latency calculation for inconsistent EMG response.

In this case as shown in Figure 4.10, the algorithm calculates the latency with the time of the first identified maxima, because is the largest. But in cases like these, other methods such as the derivative maxima, explained later, may be more useful for the user.

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
18.138	5.183	18.161	0.136	23.0

Mean latency: 23.0 ms

Figure 4.10: Latency calculation using of the local maxima method of the EMG response shown in Figure 4.9.

Local minima

For these EMG responses, we can see how the local maxima method correctly detects the minima and calculates the latency regarding the times when they happen.

For the smoother EMG response, shown in Figure 4.11 , the program detects the minimum, and correctly calculates the latency, as seen in Figure 4.12.

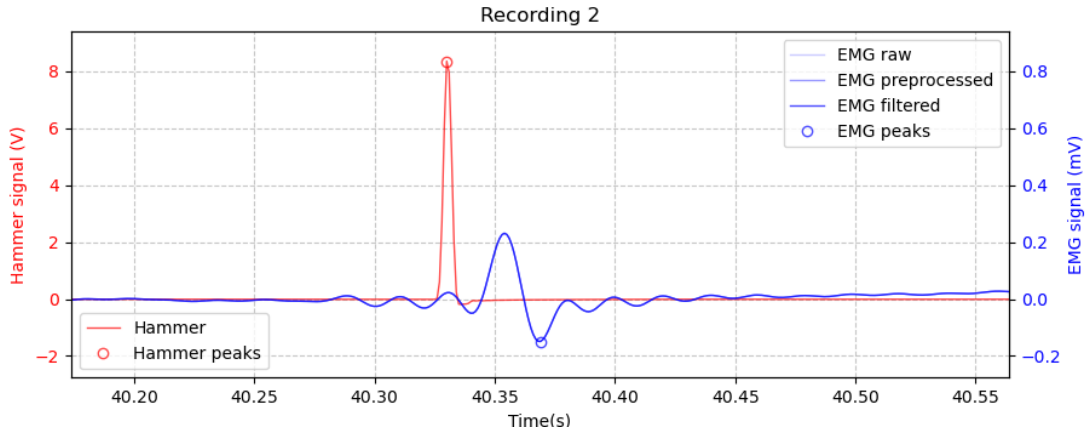

Figure 4.11: Visual representation of the local minima method of latency calculation for smooth EMG response.

```
+------------------+-----------+---------------+------------+--------------+
| Hammer peaks (s) | Value (V) | EMG peaks (s) | Value (mV) | Latency (ms) |
+------------------+-----------+---------------+------------+--------------+
|      40.33       |   8.363   |     40.369    |   -0.149   |      39.0    |
+------------------+-----------+---------------+------------+--------------+
Mean latency: 39.0 ms
```

Figure 4.12: Latency calculation using the local minima method of the EMG response shown in Figure 4.11.

For the second response, shown in Figure 4.13, the program also detects the minimum correctly and proceeds to calculate the latency accordingly, as seen in Figure 4.14.

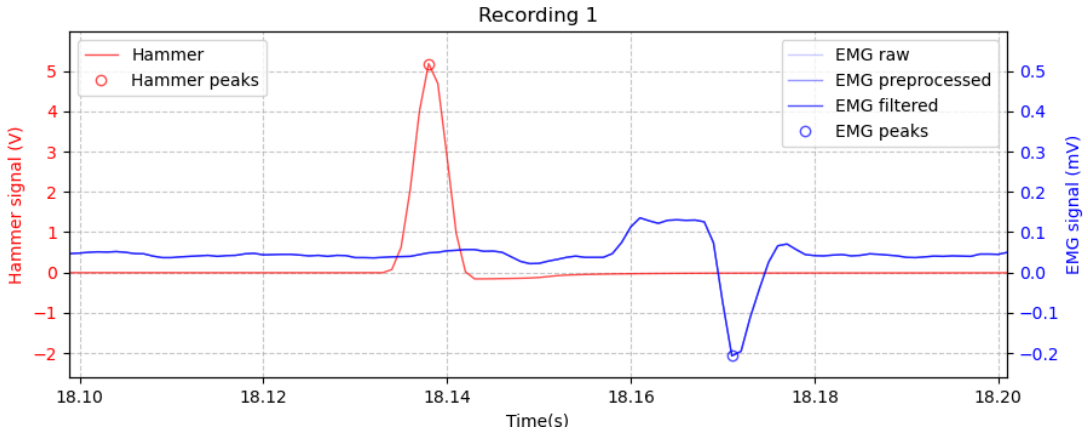

Figure 4.13: Visual representation of the local minima method of latency calculation for inconsistent EMG response.

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
18.138	5.185	18.171	-0.207	33.0

Mean latency: 33.0 ms

Figure 4.14: Latency calculation using the local minima method of the EMG response shown in Figure 4.13.

Local maxima and minima

Taking the same two EMG responses from the same recordings, it is shown that in both Figures 4.15 and 4.17, the maxima and minima are identified. The latency calculations will then be performed using the highest value among the relative extrema.

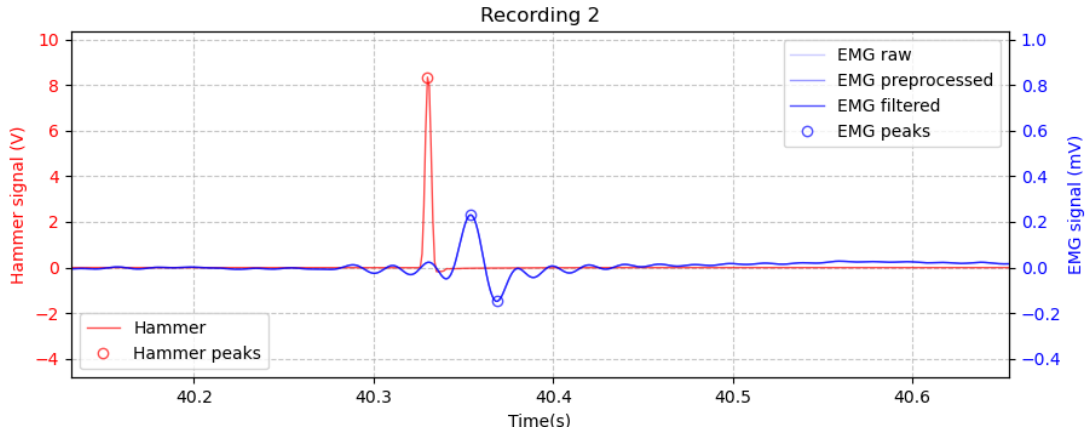

Figure 4.15: Visual representation of the local maxima and minima method of latency calculation for smooth EMG response

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
40.33	8.356	40.354	0.231	24.0

Mean latency: 24.0 ms

Figure 4.16: Latency calculation using the local maxima and minima method of the EMG response shown in Figure 4.15.

In the first case, the maximum magnitude is larger that the one of the minimum, so the latency calculation will be performed using the EMG maximum's time, as shown in Figure 4.16.

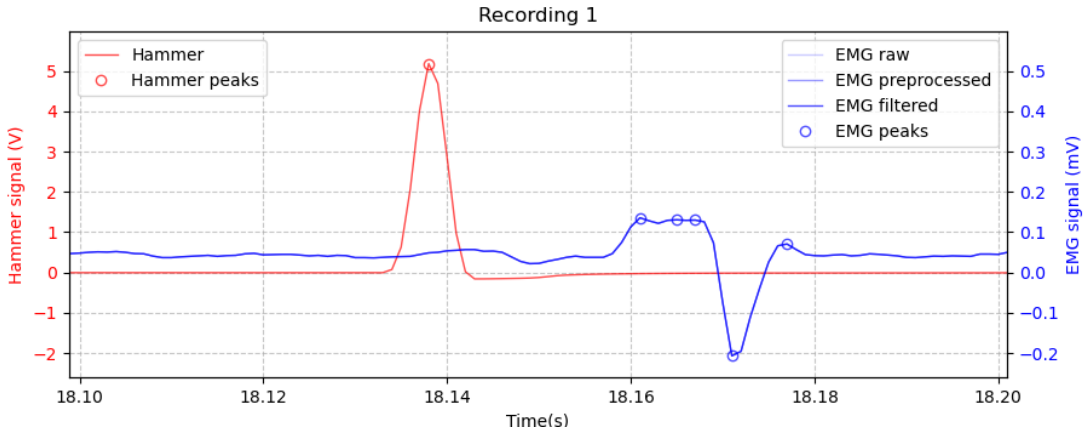

Figure 4.17: Visual representation of the local maxima and minima method of latency calculation for inconsistent EMG response.

```
+------------------+-----------+----------------+------------+--------------+
| Hammer peaks (s) | Value (V) | EMG peaks (s)  | Value (mV) | Latency (ms) |
+------------------+-----------+----------------+------------+--------------+
|      18.138      |   5.185   |     18.171     |   -0.207   |      33.0    |
+------------------+-----------+----------------+------------+--------------+
Mean latency: 33.0 ms
```

Figure 4.18: Latency calculation using the local maxima and minima method of the EMG response shown in Figure 4.17.

In the second case, the absolute value of the minimum of the EMG is larger, so it is the one that the algorithm takes into account to calculate the latency, as seen in Figure 4.18.

Derivative maxima

The last method the user can select for latency calculation is the derivative maxima option. For EMG responses like the ones seen in Figure 4.19, this method may not be useful and the local maxima method should be preferred.

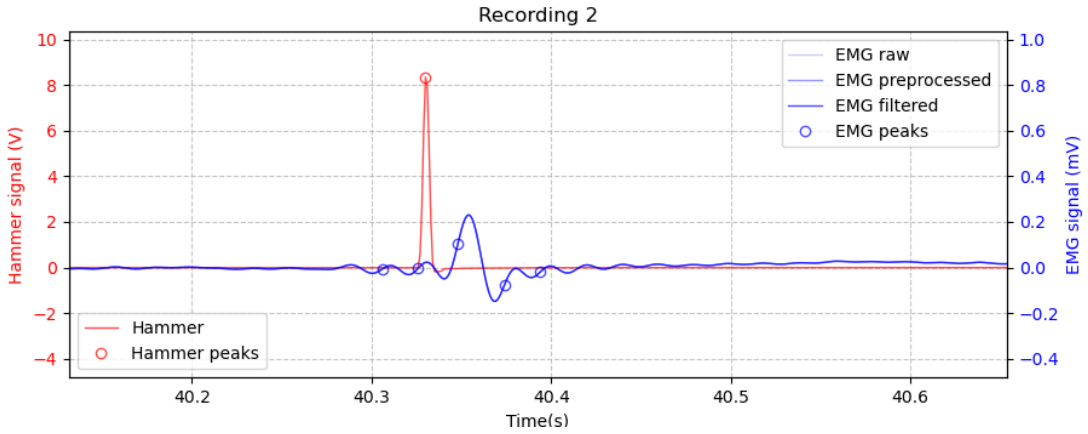

Figure 4.19: Visual representation of the derivative maxima method of latency calculation for smooth EMG response

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
40.33	8.356	40.348	0.102	18.0

Mean latency: 18.0 ms

Figure 4.20: Latency calculation using the derivative maxima method of the EMG response shown in Figure 4.19.

However, the derivative maxima latency calculation method can be useful for EMG responses like the one shown in Figure 4.21. In this kind of signals, there are more than one maximum defined, three in this case. Calculating the maxima of the derivative for the latency calculation may reduce the error when performing the calculations for entire recordings.

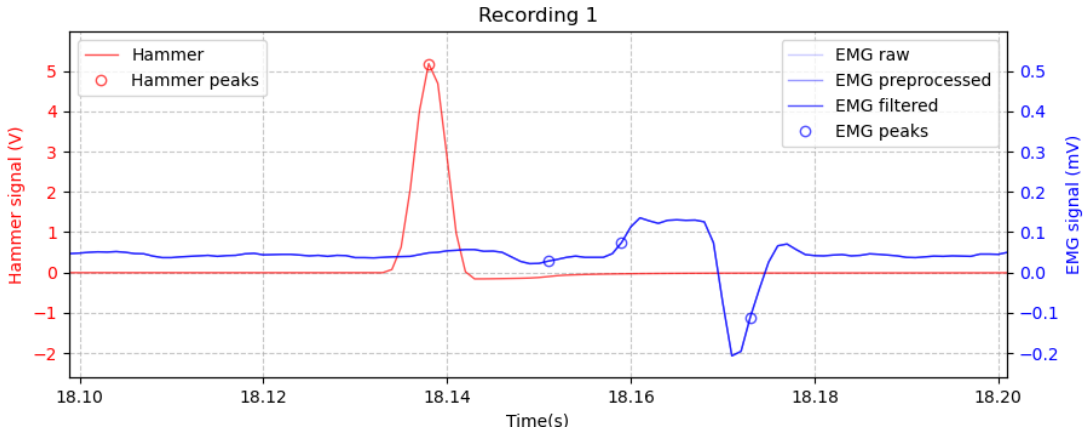

Figure 4.21: Visual representation of the derivative maxima calculation for inconsistent EMG response.

Hammer peaks (s)	Value (V)	EMG peaks (s)	Value (mV)	Latency (ms)
18.138	5.183	18.159	0.074	21.0

Mean latency: 21.0 ms

Figure 4.22: Latency calculation using the derivative maxima method of the EMG response shown in Figure 4.21.

4.2 Comparison with pre-existing solutions

In this section, a comparison between our project and previous studies and methods conducted in the field is provided, with similar objectives and methodologies.

- **Current solution 1.** Although the work described in [31] contains a GUI that allows the user to interact with an EMG signal and calculate the reflex latency response, there are a few differences between this program and ours. Among these differences is the type of reflex; we analyze the patellar reflex, while they analyze the H-reflex, or the type of stimulus; ours is mechanical and theirs is electrical.

- **Current solution 2.** Regarding the method used in [32], there are plenty of similarities, the type of reflex and stimulus analyzed. The software they use is also similar to the

one used in the URJC laboratories. The main difference was the points they used to calculate the latency: from the onset of the reflex hammer tap to the onset of the EMG response, while in the URJC laboratories they calculate them from peak to peak or peak to minimum. The derivative maximum method for latency calculations was based on this approach.

- **Current solution 3.** One of the methods used in the URJC laboratories consisted of using a spreadsheet software to perform the calculations. The extracted *.txt* files had to be opened as a spreadsheet, and the user had to manually represent them and calculate the latencies. This method has many disadvantages, such as the inability to process signals, manual and time-consuming calculations, or the increase in overall cost.

- **Current solution 4.** This method consists of using the LabChart software to calculate the latencies. After displaying the signals, the user can position the cursor over the peaks of the signals to find out the time at which they occur, and manually calculate the latency by subtracting the time of the EMG from that of the hammer. LabChart enables the display of the signals, the segmentation into hammer strokes, and the visualization of different channels simultaneously. Although LabChart has some advantages, the latency calculation, which is the main issue addressed by this project, is not automated and can lead to human error. Also LabChart is not an intuitive software, and can be difficult to use without proper training. To ensure that a researcher is adequately trained, the ADInstruments company offers training courses that can cost between 850 € and 1,000 €, which will considerably increase costs.

This project overcomes the limitations of previous solutions by providing an automated and intuitive method for calculating reflex latency responses. Unlike previous methods, our approach does not require manual calculations, which reduces the risk of human error and saves time. Additionally, our software is designed to be intuitive and easy to use, eliminating the need for extensive training or costly courses. This makes our project a more efficient and cost-effective solution for researchers in the field.

4.3 Future scope

Although this application has successfully achieved all the objectives set at the beginning, there is room for further development.

- A signal segmentation option could be useful to researchers if the study of only one reflex response is required. The signal, as a *.txt* file, could be saved after the filters have been applied or after segmentation.
- Because some of the extracted files came with defects and had to be discarded, the program could have an implementation to detect these files and automatically reject them or warn the user.
- The implementation of more filters could be added to the 'Filters' menu, maybe oriented to other types of reflexes or muscles, to broaden the possibilities of this app.
- The possibility to enable the user to input the parameters of the filters or thresholds themselves could also be implemented.
- This project is intended to be implemented as a an *.exe* for easier access to users, but this option is only available for MS Windows. A possible future improvement could be to include other operating systems such as Linux or macOs.

Chapter 5

Conclusions

The main objective of this End of Degree Project, was to design and develop a tool that efficiently and automatically calculates the reflex response latency between a reflex hammer tap and the subject's response, and it was accomplished. The following improvements were implemented with respect to the previous URJC laboratory method:

- Automatization of the patellar reflex latency calculation task.
- Development of a more reliable, faster, and intuitive tool.

Additionally, the secondary objectives were achieved:

- Design of a GUI that enables the user to visualize, adapt, and perform calculations of the selected files.
- File selection and data processing of previously extracted *.txt* files by the 'LabChart' software.
- Several visualization options, including multiple choice selection so the user has the possibility to decide which graphs to show or hide.
- Different processing options for the EMG signal, with a single choice menu that enables the user to decide which filter to apply.
- Reliable latency calculation, multiple method development and algorithm design of the selected region of the signals to correctly choose the EMG response in relation to the reflex hammer tap.

- An executable format for easier implementation at the university's laboratory.

Despite its main function, the latency calculation in tendon reflex measurements, this program allows a study of the filters and postprocessing methods usually used to smooth and prepare EMG signals. Thanks to this implementation, the following assertion could be made: the combination of the processing and calculation method, will depend on the variability of the signals, and the objectives the user has. These results can help the user, especially if they are not familiar with processing signals, to choose the appropriate filter wisely.

This project was designed with the intention of making a meaningful impact and providing a practical solution for researchers. Because necessities may arise in the future, it is envisioned that this project will serve as a foundation for future developments, being adapted and improved.

In order to accomplish this project, I had to apply the knowledge I acquired during my years studying Biomedical Engineering. *Programming Fundamentals* and *Programming in Network Environments* provided valuable skills for the development of the GUI 's code, and subjects like *Physiological Signal Processing* and *Linear Systems* were useful for the processing of the signal. Thanks to subjects like *Human Physiology*, which provided the knowledge to understand the anatomy and mechanisms of human reflexes, I was able to accurately interpret and analyze the physiological data. Overall, the integration of these various subjects allowed me to successfully complete the project and achieve meaningful results.

Working on this project has been an enriching experience, offering me substantial learning opportunities across various domains of Biomedical Engineering. This project challenged me to apply theoretical concepts to real-world problems, improving my problem-solving abilities and practical skills. Overall, this End of Degree Project has broadened my knowledge while allowing me to become a part of an actual solution to a concrete problem.

Bibliography

[1] Hall, J. E., Guyton, A. C., Hall, M. E. (2021). Tratado de fisiología médica (14ª). Elsevier.

[2] «Understanding Your Electromyography Results». Verywell Health, https://www.verywellhealth.com/understanding-emg-and-ncs-results-2488642. Accedido 11 de abril de 2024.

[3] https://www.adinstruments.com/products/powerlab/education

[4] Boes, Christopher J. «The History of Examination of Reflexes». Journal of Neurology, vol. 261, n.o 12, diciembre de 2014, pp. 2264-74. Springer Link, https://doi.org/10.1007/s00415-014-7326-7.

[5] Billman, George E. «Homeostasis: The Underappreciated and Far Too Often Ignored Central Organizing Principle of Physiology». Frontiers in Physiology, vol. 11, marzo de 2020, p. 200. PubMed Central, https://doi.org/10.3389/fphys.2020.00200.

[6] Waterhouse, James, y Iain Campbell. «Reflexes: principles and properties». Anaesthesia Intensive Care Medicine, vol. 15, n.o 4, abril de 2014, pp. 201-06. ScienceDirect, https://doi.org/10.1016/j.mpaic.2014.01.017.

[7] He, Ke, et al. «An Artificial Somatic Reflex Arc». Advanced Materials, vol. 32, n.o 4, enero de 2020, p. 1905399. DOI.org (Crossref), https://doi.org/10.1002/adma.201905399.

[8] https://senecalearning.com/en-GB/revision-notes/a-level/biology/aqa/6-1-4-reflexesreflexes-reflex-arc

[9] Ackerley, Rochelle. «Somatosensation and Body Perception: The Integration of Afferent Signals in Multisensory Cognitive Processes». Cognitive Archaeology, Body Cognition, and the Evolution of Visuospatial Perception, Elsevier, 2023, pp. 3-23. DOI.org (Crossref), https://doi.org/10.1016/B978-0-323-99193-3.00007-6.

[10] Zampieri, Niccolò, y Joriene C. de Nooij. «Regulating muscle spindle and Golgi tendon organ proprioceptor phenotypes». Current Opinion in Physiology, vol. 19, febrero de 2021, pp. 204-10. ScienceDirect, https://doi.org/10.1016/j.cophys.2020.11.001.

[11] Nick, Jan M. «Deep Tendon Reflexes: The What, Why, Where, and How of Tapping». Journal of Obstetric, Gynecologic Neonatal Nursing, vol. 32, n.o 3, mayo de 2003, pp. 297-306. DOI.org (Crossref), https://doi.org/10.1177/0884217503253491.

[12] Lin-Wei, Ooi, et al. «Deep Tendon Reflex: The Tools and Techniques. What Surgical Neurology Residents Should Know». The Malaysian Journal of Medical Sciences: MJMS, vol. 28, n.o 2, abril de 2021, pp. 48-62. PubMed Central, https://doi.org/10.21315/mjms2021.28.2.5.

[13] Dietz, V., et al. «Changes in Spinal Reflex and Locomotor Activity after a Complete Spinal Cord Injury: A Common Mechanism?» Brain, vol. 132, n.o 8, agosto de 2009, pp. 2196-205. DOI.org (Crossref), https://doi.org/10.1093/brain/awp124.

[14] Basinger, Hayden, y Jeffery P. Hogg. «Neuroanatomy, Brainstem». StatPearls, StatPearls Publishing, 2024. PubMed, http://www.ncbi.nlm.nih.gov/books/NBK544297/.

[15] Jones, J. S. «The Nottingham Medical School.» BMJ, vol. 4, n.o 5987, octubre de 1975, pp. 29-31. DOI.org (Crossref), https://doi.org/10.1136/bmj.4.5987.29.

[16] Futagi, Yasuyuki, et al. «The Grasp Reflex and Moro Reflex in Infants: Hierarchy of Primitive Reflex Responses». International Journal of Pediatrics, vol. 2012, 2012, pp. 1-10. DOI.org (Crossref), https://doi.org/10.1155/2012/191562.

[17] Brown, Josephine V., y W. Timm Fredrickson. «The Relationship between Sucking and Grasping in the Human Newborn: A Precursor of Hand-mouth Coordination?» Developmental Psychobiology, vol. 10, n.o 6, noviembre de 1977, pp. 489-98. DOI.org (Crossref), https://doi.org/10.1002/dev.420100602.

[18] Morrow, Jasper M., y Mary M. Reilly. «The Babinski Sign». British Journal of Hospital Medicine, vol. 72, n.o Sup10, octubre de 2011, pp. M157-59. DOI.org (Crossref), https://doi.org/10.12968/hmed.2011.72.Sup10.M157.

[19] Santuz, Alessandro, y Turgay Akay. «Muscle Spindles and Their Role in Maintaining Robust Locomotion». The Journal of Physiology, vol. 601, n.o 2, enero de 2023, pp. 275-85. DOI.org (Crossref), https://doi.org/10.1113/JP282563.

[20] Rodriguez-Beato, Freddie Y., y Orlando De Jesus. «Physiology, Deep Tendon Reflexes». StatPearls, StatPearls Publishing, 2024. PubMed, http://www.ncbi.nlm.nih.gov/books/NBK562238/.

[21] De Luca, Carlo. «Electromyography». Encyclopedia of Medical Devices and Instrumentation, editado por John G. Webster, 1.a ed., Wiley, 2006. DOI.org (Crossref), https://doi.org/10.1002/0471732877.emd097.

[22] Rubin, Devon I. «Needle Electromyography: Basic Concepts». Handbook of Clinical Neurology, vol. 160, Elsevier, 2019, pp. 243-56. DOI.org (Crossref), https://doi.org/10.1016/B978-0-444-64032-1.00016-3.

[23] Omar, Abdillahi, et al. «Physiology, Neuromuscular Junction». StatPearls, StatPearls Publishing, 2024. PubMed, http://www.ncbi.nlm.nih.gov/books/NBK470413/.

[24] Neuromuscular junction and transmission - Labster. https://theory.labster.com/muscle-contraction/. Accedido 16 de mayo de 2024.

[25] Cavalcanti Garcia M, , M. Vieira T. Surface electromyography: Why, when and how to use it. Revista Andaluza de Medicina del Deporte [Internet]. 2011;4(1):17-28. Recuperado de: https://www.redalyc.org/articulo.oa?id=323327665004

[26] Drost, Gea, et al. «Clinical Applications of High-Density Surface EMG: A Systematic Review». Journal of Electromyography and Kinesiology, vol. 16, n.o 6, diciembre de 2006, pp. 586-602. DOI.org (Crossref), https://doi.org/10.1016/j.jelekin.2006.09.005.

[27] Pilkar, Rakesh, et al. «Use of Surface EMG in Clinical Rehabilitation of Individuals With SCI: Barriers and Future Considerations». Frontiers in Neurology, vol. 11, diciembre de 2020, p. 578559. PubMed Central, https://doi.org/10.3389/fneur.2020.578559.)

[28] Lanska, Douglas J. «The History of Reflex Hammers». Neurology, vol. 39, n.o 11, noviembre de 1989, pp. 1542-1542. DOI.org (Crossref), https://doi.org/10.1212/WNL.39.11.1542.

[29] Marshall, Garrett L., y James W. Little. «Deep Tendon Reflexes: A Study Of Quantitative Methods». The Journal of Spinal Cord Medicine, vol. 25, n.o 2, julio de 2002, pp. 94-99. DOI.org (Crossref), https://doi.org/10.1080/10790268.2002.11753608.

[30] Discover the World of Reflex Hammers: Types, Uses, and Leading Manufacturers. https://www.indosurgicals.com/page/Reflex-Hammers.php. Accedido 23 de abril de 2024.

[31] Moukarzel, George, et al. «A MATLAB Application for Automated H-Reflex Measurements and Analyses». Biomedical Signal Processing and Control, vol. 66, abril de 2021, p. 102448. DOI.org (Crossref), https://doi.org/10.1016/j.bspc.2021.102448.

[32] Haen Whitmer, K.M. (2021). A Mixed Course-Based Research Approach to Human Physiology. Ames, IA: Iowa State University Digital Press. https://iastate.pressbooks.pub/curehumanphysiology/

[33] https://www.python.org/

[34] Krasner, Glenn E., y Stephen T. Pope. «A cookbook for using the model-view controller user interface paradigm in Smalltalk-80». Journal of Object-Oriented Programming, vol. 1, n.o 3, agosto de 1988, pp. 26-49.

[35] Raez, M. B. I., et al. «Techniques of EMG signal analysis: detection, processing, classification and applications». Biological Procedures Online, vol. 8, marzo de 2006, pp. 11-35. PubMed Central, https://doi.org/10.1251/bpo115.

[36] Schafer, Ronald. «What Is a Savitzky-Golay Filter? [Lecture Notes]». IEEE Signal Processing Magazine, vol. 28, n.o 4, julio de 2011, pp. 111-17. DOI.org (Crossref), https://doi.org/10.1109/MSP.2011.941097.

[37] Frijns, C. J. M., et al. «Normal Values of Patellar and Ankle Tendon Reflex Latencies». Clinical Neurology and Neurosurgery, vol. 99, n.o 1, febrero de 1997, pp. 31-36. DOI.org (Crossref), https://doi.org/10.1016/S0303-8467(96)00593-8.

Additional Material

Python code

```
1
2  import sys
3  import tkinter as tk
4
5  import matplotlib.backend_bases
6  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
7  from tabulate import tabulate
8
9  import matplotlib.pyplot as plt
10 import numpy as np
11 # from scipy import signal as sp
12 import scipy as sp
13
14 import pyemgpipeline
15
16
17 class LcaFileUtils:
18     @staticmethod
19     def ask_open_txt_file_name():
20         file_name = tk.filedialog.askopenfilename(defaultextension=".txt",
21                                                   filetypes=[("Text files", "*.txt"),
22                                                              ("All files", "*.*")])
23         return file_name
24
25     @staticmethod
```

```
26      def ask_save_txt_file_name():
27          file_name = tk.filedialog.asksaveasfilename(defaultextension=".txt",
28                                                      filetypes=[("Text files",
    "*.txt"),
29                                                                 ("All files", "*.*")])
30          return file_name
31
32      @staticmethod
33      def read(file_name):
34          time_data = []
35          hammer_raw = []
36          emg_raw = []
37          reading = False
38          with open(file_name, 'r') as file:
39              for line in file:
40                  if reading:
41                      try:
42                          values = [float(value.replace(',', '.')) for value in
    line.split('\t')]
43                          time_data.append(values[0])
44                          hammer_raw.append(values[1])
45                          emg_raw.append(values[2])
46                      except ValueError:
47                          break
48                  elif line.startswith('0\t'):
49                      reading = True
50          return time_data, hammer_raw, emg_raw
51
52
53  # static methods for signal handling
54  class LcaSignalUtils:
55      @staticmethod
56      def find_peaks(xdata, ydata_post, ydata_src, xlim, height):
57          # Find indices in range
58          if xlim is None:
59              xrange = xdata
60              yrange_post = ydata_post
```

```
61              yrange_src = ydata_src
62          else:
63              irange = np.where((xdata >= xlim[0]) & (xdata <= xlim[1]))[0]
64              # Find x's and y's in range
65              xrange = np.array(xdata)[irange]
66              yrange_post = np.array(ydata_post)[irange]
67              yrange_src = np.array(ydata_src)[irange]
68
69          # Find only peaks with a minimum height
70          ipeaks, _ = sp.signal.find_peaks(yrange_post, height=height)
71
72          xpeaks = [round(xrange[i], 3) for i in ipeaks]
73          ypeaks_post = [round(yrange_post[i], 3) for i in ipeaks]
74          ypeaks_src = [round(yrange_src[i], 3) for i in ipeaks]
75          return xpeaks, ypeaks_post, ypeaks_src
76
77      @staticmethod
78      def find_hammer_peaks(time_data, hammer, xlim):
79          height = 0.1
80          peak_times_hammer, peak_values_hammer, _ =
    LcaSignalUtils.find_peaks(time_data, hammer, hammer, xlim, height)
81          return peak_times_hammer, peak_values_hammer
82
83      @staticmethod
84      def find_emg_peaks(time, emg, latency_postprocess_name, xlim):
85          emg_postprocessed = LcaSignalUtils.postprocess_emg_filtered(emg,
    latency_postprocess_name)
86          if latency_postprocess_name == 'gradient':
87              height = 0.005
88          else:
89              height = 0.06
90          peak_times_emg, peak_values_emg_post, peak_values_emg_src =
    LcaSignalUtils.find_peaks(time, emg_postprocessed,
91
                  emg,
92
                  xlim, height)
```

```
93          return peak_times_emg, peak_values_emg_post, peak_values_emg_src
94
95      ############################################################
96
97      # DEFAULT_HAMMER_FILTER_NAME = '-'
98      # DEFAULT_EMG_FILTER_NAME = 'rolling_rms'
99      DEFAULT_EMG_FILTER_NAME = '-'
100     DEFAULT_LATENCY_POSTPROCESS_NAME = 'max'
101
102     ############################################################
103
104     SAMPLING_FREQ = 1000 # Hz
105     NYQUIST_FREQ = SAMPLING_FREQ / 2
106
107     @staticmethod
108     def normalize_frequency(freq):
109         return freq / LcaSignalUtils.NYQUIST_FREQ
110
111     @staticmethod
112     def build_digital_freqs(btype):
113         if btype == 'lowpass':
114             freqs = LcaSignalUtils.LOW_CUTOFF_FREQ
115         elif btype == 'highpass':
116             freqs = LcaSignalUtils.HIGH_CUTOFF_FREQ
117         elif btype == 'bandpass':
118             freqs = (LcaSignalUtils.LOW_CUTOFF_FREQ,
    LcaSignalUtils.HIGH_CUTOFF_FREQ)
119         return freqs
120
121     def build_digital_normalized_freqs(btype):
122         if btype == 'lowpass':
123             normalized_freqs =
    LcaSignalUtils.normalize_frequency(LcaSignalUtils.LOW_CUTOFF_FREQ)
124         elif btype == 'highpass':
125             normalized_freqs =
    LcaSignalUtils.normalize_frequency(LcaSignalUtils.HIGH_CUTOFF_FREQ)
126         elif btype == 'bandpass':
```

```
127             normalized_freqs = (
128 
        LcaSignalUtils.normalize_frequency(LcaSignalUtils.LOW_CUTOFF_FREQ),
129 
        LcaSignalUtils.normalize_frequency(LcaSignalUtils.HIGH_CUTOFF_FREQ))
130         return normalized_freqs
131 
132     ############################################################
133 
134     # @staticmethod
135     # def filter_hammer(hammer, hammer_filter_name):
136     #     if hammer_filter_name == '-':
137     #         return hammer
138     #     elif hammer_filter_name == 'default':
139     #         return LcaSignalUtils.filter_hammer(hammer,
        LcaSignalUtils.DEFAULT_HAMMER_FILTER_NAME)
140     #     else:
141     #         return None
142 
143     ############################################################
144 
145     # @staticmethod
146     # def rolling_rms_filter(signal, half_window_size):
147     #     window_size = 2 * half_window_size + 1
148     #     window = np.ones(window_size) / float(window_size)
149     #
150     #     return np.sqrt(
151     #         sp.signal.fftconvolve(
152     #             np.power(signal, 2),
153     #             window,
154     #             'same'))
155 
156     # @staticmethod
157     # def filter_rolling_rms(signal):
158     #     return LcaSignalUtils.rolling_rms_filter(signal, 2)
159     #     # return LcaSignalUtils.rolling_rms_filter(signal, 3)
160     #     # return LcaSignalUtils.rolling_rms_filter(rolling_rms_filter(signal, 3), 1)
```

```
161
162     ############################################################
163
164     FILTER_ORDER = 4
165
166     HIGH_CUTOFF_FREQ = 60  # Hz
167     LOW_CUTOFF_FREQ = 30  # Hz
168
169     STOP_ATTENUATION = 40  # dB
170
171     @staticmethod
172     def  build_digital_ba_butter_filter  (btype):
173         b, a = sp.signal.butter(LcaSignalUtils.FILTER_ORDER,
        LcaSignalUtils.build_digital_normalized_freqs(btype),
174                                 btype=btype, analog=False, output='ba')
175         return b, a
176
177     # @staticmethod
178     # def  build_digital_sos_butter_filter  (btype):
179     #     sos = sp.signal.butter(LcaSignalUtils.FILTER_ORDER,
        LcaSignalUtils.build_digital_normalized_freqs(btype),
180     #                            btype=btype, analog=False, output='sos')
181     #     return sos
182
183     @staticmethod
184     def  filter_butter_lowpass_filtfilt  (signal):
185         b, a = LcaSignalUtils.  build_digital_ba_butter_filter  ('lowpass')
186          filtered = sp.signal.  filtfilt  (b, a, signal)
187         return  filtered
188
189     @staticmethod
190     def  filter_butter_highpass_filtfilt  (signal):
191         b, a = LcaSignalUtils.  build_digital_ba_butter_filter  ('highpass')
192          filtered = sp.signal.  filtfilt  (b, a, signal)
193         return  filtered
194
195     @staticmethod
```

```
196     def filter_butter_bandpass_filtfilt(signal):
197         b, a = LcaSignalUtils.build_digital_ba_butter_filter('lowpass')
198         filtered = sp.signal.filtfilt(b, a, signal)
199         return filtered
200
201     # @staticmethod
202     # def filter_butter_lowpass_lfilter(signal):
203     #     b, a = LcaSignalUtils.build_digital_ba_butter_filter('lowpass')
204     #     filtered = sp.signal.lfilter(b, a, signal)
205     #     return filtered
206
207     ################################################################
208
209     # @staticmethod
210     # def build_digital_ba_cheby2_filter(btype):
211     #     b, a = sp.signal.cheby2(LcaSignalUtils.FILTER_ORDER,
        LcaSignalUtils.STOP_ATTENUATION,
212     #
        LcaSignalUtils.build_digital_normalized_freqs(btype),
213     #                                 btype=btype, analog=False, output='ba')
214     #     return b, a
215     #
216     # @staticmethod
217     # def filter_cheby2_lowpass_filtfilt(signal):
218     #     b, a = LcaSignalUtils.build_digital_ba_cheby2_filter('lowpass')
219     #     filtered = sp.signal.filtfilt(b, a, signal)
220     #     return filtered
221
222     ################################################################
223
224     @staticmethod
225     def filter_savgol(signal):
226         # return sp.signal.savgol_filter(signal, window_length=45, polyorder=4)
227         return sp.signal.savgol_filter(signal, window_length=11, polyorder=3)
228
229     ################################################################
230
```

```
231     # @staticmethod
232     # def filter_conventional (signal):
233     #     lowpass = 6
234     #     sfreq = 2000
235     #     high_band = 10
236     #     low_band = 35
237     #     amplification_factor = 2
238     #
239     #     # normalise cut-off frequencies to sampling frequency
240     #     high_band = high_band / (sfreq / 2)
241     #     low_band = low_band / (sfreq / 2)
242     #
243     #     # create bandpass filter for EMG
244     #     b1, a1 = sp.signal.butter(4, [high_band, low_band], btype='bandpass')
245     #
246     #     # process EMG signal: filter EMG
247     #     emg_filtered = sp.signal.filtfilt(b1, a1, signal)
248     #
249     #     # process EMG signal: rectify
250     #     emg_rectified = abs(emg_filtered)
251     #
252     #     # create lowpass filter and apply to rectified signal to get EMG envelope
253     #     lowpass = lowpass / (sfreq / 2)
254     #     b2, a2 = sp.signal.butter(4, lowpass, btype='lowpass')
255     #     emg_envelope = sp.signal.filtfilt(b2, a2, emg_rectified)
256     #
257     #     emg_amplified = emg_envelope * amplification_factor
258     #
259     #     return emg_amplified
260
261     ###################################################################
262
263     NOTCH_FREQ = 50.0  # Frequency to be removed (notch frequency)
264     NOTCH_BANDWIDTH = 2.0  # Bandwidth of the notch in Hz
265     NOTCH_ATTENUATION = 20  # Desired attenuation at 50 Hz in dB
266
267     # @staticmethod
```

```
268     # def filter_notch_filtfilt (signal):
269     #     Q = LcaSignalUtils.NOTCH_FREQ / LcaSignalUtils.NOTCH_BANDWIDTH
270     #     # Design the notch filter
271     #     b, a = sp.signal.iirnotch(LcaSignalUtils.NOTCH_FREQ, Q,
        LcaSignalUtils.SAMPLING_FREQ)
272     #     # Adjust the gain to achieve the desired attenuation
273     #     gain = 10 ** (-LcaSignalUtils.NOTCH_ATTENUATION / 20)
274     #     b *= gain
275     #
276     #     # filtered_emg = sp.signal.lfilter(b, a, signal)
277     #     filtered_emg = sp.signal.filtfilt(b, a, signal)
278     #
279     #     return filtered_emg
280
281     # @staticmethod
282     # def filter_pyemgpipeline(signal):
283     #     # Ensure the input is a NumPy array
284     #     if not isinstance(signal, np.ndarray):
285     #         signal = np.array(signal)
286     #
287     #     emg_trial = pyemgpipeline.wrappers.EMGMeasurement(signal, hz=1000)
288     #
289     #     # emg_trial.remove_dc_offset()
290     #
291     #     # emg_trial.apply_bandpass_filter(lowcut=20, highcut=500, order=4)
292     #
293     #     # emg_trial.apply_full_wave_rectification()
294     #
295     #     # TypeError: EMGMeasurement.apply_linear_envelope() got an unexpected
        keyword argument 'lowcut'
296     #     emg_trial.apply_linear_envelope(lowcut=6, order=2)
297     #
298     #     return emg_trial
299
300     @staticmethod
301     def filter(signal, filter_name):
302         if filter_name == '-':
```

```
303             return signal
304         # elif filter_name == 'default':
305         #     return LcaSignalUtils.filter(signal,
    LcaSignalUtils.DEFAULT_EMG_FILTER_NAME)
306         elif filter_name == 'savgol':
307             return LcaSignalUtils.filter_savgol(signal)
308         # elif filter_name == 'rolling_rms':
309         #     return LcaSignalUtils.filter_rolling_rms(signal)
310         elif filter_name == 'butter_lowpass_filtfilt':
311             return LcaSignalUtils.filter_butter_lowpass_filtfilt(signal)
312         elif filter_name == 'butter_highpass_filtfilt':
313             return LcaSignalUtils.filter_butter_highpass_filtfilt(signal)
314         elif filter_name == 'butter_bandpass_filtfilt':
315             return LcaSignalUtils.filter_butter_bandpass_filtfilt(signal)
316         # elif filter_name == 'butter_lowpass_lfilter':
317         #     return LcaSignalUtils.filter_butter_lowpass_lfilter(signal)
318         # elif filter_name == 'cheby2_lowpass_filtfilt':
319         #     return LcaSignalUtils.filter_cheby2_lowpass_filtfilt(signal)
320         # elif filter_name == 'notch_filtfilt':
321         #     return LcaSignalUtils.filter_notch_filtfilt(signal)
322         # elif filter_name == 'conventional':
323         #     return LcaSignalUtils.filter_conventional(signal)
324         # elif filter_name == 'pyemgpipeline':
325         #     return LcaSignalUtils.filter_pyemgpipeline(signal)
326         else:
327             return None
328
329     @staticmethod
330     def postprocess_emg_filtered(emg_filtered, postprocess_emg_name):
331         if postprocess_emg_name == 'max':
332             return emg_filtered
333         if postprocess_emg_name == 'min':
334             return np.multiply(emg_filtered, -1.0)
335         if postprocess_emg_name == 'max+min':
336             return np.abs(emg_filtered)
337         if postprocess_emg_name == 'gradient':
338             return np.gradient(emg_filtered)
```

```
339
340
341 class LcaData:
342     def __init__(self, file_name):
343         self.file_name = file_name
344         self.name = file_name.split('/')[-1].replace('.txt', '')
345
346         self.time_data = None
347
348         self.hammer_raw = None
349         # self.hammer_filtered = None
350         self.peak_times_hammer = None
351         self.peak_values_hammer = None
352
353         self.emg_raw = None
354         self.emg_preprocessed = None
355         self.emg_filtered = None
356         self.peak_times_emg = None
357         self.peak_values_emg = None
358
359         self.time_data, self.hammer_raw, self.emg_raw = LcaFileUtils.read(file_name)
360
361         self.emg_preprocessed = self.emg_raw
362
363         # self.set_hammer_filter(LcaSignalUtils.DEFAULT_HAMMER_FILTER_NAME)
364         self.peak_times_hammer, self.peak_values_hammer =
    LcaSignalUtils.find_hammer_peaks(
365             self.time_data, self.hammer_raw, None)
366
367         self.emg_filter_name = LcaSignalUtils.DEFAULT_EMG_FILTER_NAME
368         self.latency_postprocess_name =
    LcaSignalUtils.DEFAULT_LATENCY_POSTPROCESS_NAME
369         self.set_emg_filter(self.emg_filter_name)
370
371     # def set_hammer_filter(self, hammer_filter_name):
372     #     self.hammer_filtered = LcaSignalUtils.filter_hammer(self.hammer_raw,
    hammer_filter_name)
```

```
373     #       self.peak_times_hammer, self.peak_values_hammer =
        LcaSignalUtils.find_hammer_peaks(
374     #           self.time_data, self.hammer_filtered, None)
375
376     def set_emg_filter(self, emg_filter_name):
377         self.emg_filter_name = emg_filter_name
378         self.emg_filtered = LcaSignalUtils.filter(self.emg_preprocessed,
        emg_filter_name)
379
380         self.set_latency_postprocess(self.latency_postprocess_name)
381
382     def set_latency_postprocess(self, latency_postprocess_name):
383         self.latency_postprocess_name = latency_postprocess_name
384         self.peak_times_emg, _, self.peak_values_emg \
385             = LcaSignalUtils.find_emg_peaks(self.time_data, self.emg_filtered,
        self.latency_postprocess_name, None)
386
387     def reset_emg_preprocessed(self):
388         self.emg_preprocessed = self.emg_raw
389         self.set_emg_filter(self.emg_filter_name)
390
391     def apply_current_emg_filter(self):
392         self.emg_preprocessed = self.emg_filtered
393         self.set_emg_filter('-')
394
395     def calc_stats_text(self, xlim):
396         peak_times_hammer, peak_values_hammer = LcaSignalUtils.find_hammer_peaks(
397             self.time_data, self.hammer_raw, xlim)
398         peak_times_emg, peak_values_emg_post, peak_values_emg_src =
        LcaSignalUtils.find_emg_peaks(
399             self.time_data, self.emg_filtered, self.latency_postprocess_name, xlim)
400
401         def calc_latency_row(peak_time_hammer, peak_value_hammer):
402             latency_row = (peak_time_hammer, peak_value_hammer, None, None, None)
403             max_peak_values_emg_post = -10000
404             for i, peak_time_emg in enumerate(peak_times_emg):
405                 # only in range 0 to 500 ms
```

```
406                     if peak_time_hammer < peak_time_emg and peak_time_emg <=
    peak_time_hammer + 0.5:
407                         if peak_values_emg_post[i] > max_peak_values_emg_post:
408                             latency_row = (peak_time_hammer,
409                                            peak_value_hammer,
410                                            peak_time_emg,
411                                            peak_values_emg_src[i],
412                                            round((peak_time_emg - peak_time_hammer) *
    1000, 0))
413                             max_peak_values_emg_post = peak_values_emg_post[i]
414             return latency_row
415
416         latency_table = [calc_latency_row(peak_time_hammer, peak_values_hammer[i])
417                          for i, peak_time_hammer in enumerate(peak_times_hammer)]
418
419         stats_text = tabulate({
420             "Hammer peaks (s)": list(map(lambda row: row[0], latency_table)),
421             "Value (V)": list(map(lambda row: row[1], latency_table)),
422             "EMG peaks (s)": list(map(lambda row: row[2], latency_table)),
423             "Value (mV)": list(map(lambda row: row[3], latency_table)),
424             "Latency (ms)": list(map(lambda row: row[4], latency_table))
425         }, headers='keys', tablefmt='pretty', showindex=False)
426
427         valid_latencies = list(
428             filter(lambda latency: latency is not None, list(map(lambda latency_row:
    latency_row[4], latency_table))))
429         mean_latency = np.mean(valid_latencies)
430         if not np.isnan(mean_latency):
431             stats_text += f"\nMean latency: {mean_latency:.1f} ms"
432
433         return stats_text
434
435
436 class LcaPlot:
437     def __init__(self, lca_data):
438         self.lca_data = lca_data
439
```

```
440         # self.figure, self.base_axes = plt.subplots(figsize=(10, 3.7))
441         # self.hammer_axes = self.base_axes.twinx()
442         # self.emg_axes = self.hammer_axes.twinx()
443
444         self.figure, self.hammer_axes = plt.subplots(figsize=(10, 3.7))
445         self.emg_axes = self.hammer_axes.twinx()
446
447         self.hammer_axes.set_title(lca_data.name)
448         self.hammer_axes.grid(True, linestyle='--', alpha=0.7)
449         self.hammer_axes.set_xlabel("Time(s)")
450         self.hammer_axes.set_ylabel("Hammer signal (V)", color='red')
451         self.hammer_axes.tick_params(axis='y', labelcolor='red')
452
453         self.emg_axes.set_ylabel("EMG signal (mV)", color='blue')
454         self.emg_axes.tick_params(axis='y', labelcolor='blue')
455
456         self.hammer_raw_plot, = self.hammer_axes.plot(self.lca_data.time_data,
    self.lca_data.hammer_raw,
457                                                       label="Hammer", color='red',
    linewidth=1, alpha=0.7)
458         # self.hammer_filtered_plot, = self.hammer_axes.plot(self.lca_data.time_data,
    self.lca_data.hammer_filtered,
459         #                                                    label="Hammer",
    color='red', linewidth=1, alpha=0.7)
460         self.hammer_peaks_plot, =
    self.hammer_axes.plot(self.lca_data.peak_times_hammer,
461
    self.lca_data.peak_values_hammer,
462                                                           'ro', markerfacecolor='none',
    markeredgecolor='red',
463                                                           label="Hammer peaks",
    alpha=0.7)
464
465         self.emg_raw_plot, = self.emg_axes.plot(self.lca_data.time_data,
    self.lca_data.emg_raw,
466                                                 label="EMG raw", color='blue',
    linewidth=1, alpha=0.3)
```

```
467         self.emg_preprocessed_plot, = self.emg_axes.plot(self.lca_data.time_data,
        self.lca_data.emg_preprocessed,
468                                                          label="EMG applied filters",
        color='blue', linewidth=1, alpha=0.3)
469         self.emg_filtered_plot, = self.emg_axes.plot(self.lca_data.time_data,
        self.lca_data.emg_filtered,
470                                                      label="EMG filtered",
        color='blue', linewidth=1.2, alpha=0.8)
471         self.emg_peak_plot, = self.emg_axes.plot(self.lca_data.peak_times_emg,
        self.lca_data.peak_values_emg,
472                                                  'bo', markerfacecolor='none',
        markeredgecolor='blue',
473                                                  label="EMG peaks", alpha=0.7)
474
475         self.initial_xlim = self.emg_axes.get_xlim()
476         ylim_hammer = self.hammer_axes.get_ylim()
477         ylim_emg = self.emg_axes.get_ylim()
478         self.initial_ylim = (min(ylim_hammer[0] / 10, ylim_emg[0]),
        max(ylim_hammer[1] / 10, ylim_emg[1]))
479
480         # self.time_range = matplotlib.widgets.RangeSlider(self.base_axes,
        "Selection", self.initial_xlim[0], self.initial_xlim[1])
481         ## self.time_range.set_min(self.initial_xlim[0])
482         ## self.time_range.set_max(self.initial_xlim[1])
483         # self.time_range.set_val((self.initial_xlim[0], self.initial_xlim[1]))
484
485         self.reset_zoom()
486
487         # self.freq_axes = self.figure.add_axes([0.20, 0.1, 0.60, 0.03])
488
489     def reset_zoom(self):
490         # self.base_axes.set_xlim(self.initial_xlim)
491         self.hammer_axes.set_xlim(self.initial_xlim)
492         self.emg_axes.set_xlim(self.initial_xlim)
493
494         # self.base_axes.set_ylim(self.initial_ylim[0], self.initial_ylim[1])
495         self.hammer_axes.set_ylim(self.initial_ylim[0] * 10, self.initial_ylim[1] *
```

```
        10)
496             self.emg_axes.set_ylim(self. initial_ylim )
497
498         def set_emg_filter (self, emg_filter_name):
499             self.lca_data. set_emg_filter (emg_filter_name)
500
501             self. emg_filtered_plot .set_ydata(self.lca_data.emg_filtered)
502             self.emg_peak_plot.set_xdata(self.lca_data.peak_times_emg)
503             self.emg_peak_plot.set_ydata(self.lca_data.peak_values_emg)
504
505         def set_latency_postprocess (self, latency_postprocess_name):
506             self.lca_data. set_latency_postprocess (latency_postprocess_name)
507
508             self.emg_peak_plot.set_xdata(self.lca_data.peak_times_emg)
509             self.emg_peak_plot.set_ydata(self.lca_data.peak_values_emg)
510
511         def reset_emg_preprocessed(self):
512             self.lca_data.reset_emg_preprocessed()
513
514             self.emg_preprocessed_plot.set_ydata(self.lca_data.emg_preprocessed)
515             self. emg_filtered_plot .set_ydata(self.lca_data.emg_filtered)
516             self.emg_peak_plot.set_xdata(self.lca_data.peak_times_emg)
517             self.emg_peak_plot.set_ydata(self.lca_data.peak_values_emg)
518
519         def apply_current_emg_filter (self):
520             self.lca_data. apply_current_emg_filter ()
521
522             self.emg_preprocessed_plot.set_ydata(self.lca_data.emg_preprocessed)
523             self. emg_filtered_plot .set_ydata(self.lca_data.emg_filtered)
524             self.emg_peak_plot.set_xdata(self.lca_data.peak_times_emg)
525             self.emg_peak_plot.set_ydata(self.lca_data.peak_values_emg)
526
527
528 class LcaPlotWindow:
529
530         def __init__(self, lca_data):
531             self.lca_data = lca_data
```

```
532         self . lca_plot  = LcaPlot( lca_data )
533
534         # Cannot create  variables  before  root_window is  created
535         self .root_window = tk.Tk()
536         self .canvas  = None
537
538         self .show_hammer_raw_plot = tk.IntVar(master= self .root_window, value=1)
539         #  self .show_hammer_filtered_plot  = tk. IntVar (master= self . root_window,
    value=0)
540         self .show_hammer_peaks_plot = tk.IntVar(master= self .root_window, value=1)
541         self .show_emg_raw_plot = tk.IntVar(master= self .root_window, value=1)
542         self .show_emg_preprocessed_plot = tk. IntVar (master= self .root_window, value=1)
543         self . show_emg_filtered_plot  = tk . IntVar (master= self .root_window, value=1)
544         self .show_emg_peaks_plot = tk.IntVar (master= self .root_window, value=1)
545
546         self .emg_filter_name  = tk . StringVar (master= self .root_window,
    value=LcaSignalUtils .DEFAULT_EMG_FILTER_NAME)
547
548         self . latency_postprocess_name  = tk . StringVar (master= self .root_window,
549
    value=LcaSignalUtils .DEFAULT_LATENCY_POSTPROCESS_NAME)
550
551         self .create_root_window( self .root_window)
552
553         self .  refresh_plot  ()
554
555     def  create_root_window( self ,  root_window):
556         root_window. title ("EMG latency  calculation  app")
557         root_window.iconbitmap(r"lca . ico")
558
559         menubar =  self .create_menubar(root_window)
560         root_window.config(menu=menubar)
561
562         # frame  = tk . Frame(window)
563         # frame.pack()
564         frame  = tk . Frame(root_window)
565         frame.pack( fill =tk . BOTH, expand=True)
```

```
566
567         # scrollbar_x = tk.Scrollbar(frame, orient="horizontal",
    command=self.lca_plot.hammer_axes.set_xlim)
568         # scrollbar_x.pack(side=tk.BOTTOM, fill=tk.X)
569         #
570         # scrollbar_y = tk.Scrollbar(frame, orient="vertical",
    command=self.lca_plot.hammer_axes.set_ylim)
571         # scrollbar_y.pack(side=tk.RIGHT, fill=tk.Y)
572
573         # bottom_frame = tk.Frame(window)
574         # bottom_frame.pack(side=tk.BOTTOM)
575
576         self.canvas = self.create_canvas(frame)
577         canvas_widget = self.canvas.get_tk_widget()
578         canvas_widget.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
579
580         toolbar = self.create_toolbar(self.canvas, frame)
581
582         return root_window
583
584     def create_menubar(self, parent):
585         menubar = tk.Menu(parent)
586
587         file_menu = self.create_file_menu(menubar)
588         menubar.add_cascade(label="File", menu=file_menu)
589
590         view_menu = self.create_view_menu(menubar)
591         menubar.add_cascade(label="View", menu=view_menu)
592
593         # emg_preprocess_menu = self.create_emg_preprocess_menu(menubar)
594         # menubar.add_cascade(label="User-applied filters",
    menu=emg_preprocess_menu)
595
596         emg_filter_menu = self.create_emg_filter_menu(menubar)
597         menubar.add_cascade(label="Filters", menu=emg_filter_menu)
598
599         latency_menu = self.create_latency_menu(menubar)
```

```
600         menubar.add_cascade(label="Latency", menu=latency_menu)
601
602         return menubar
603
604     def create_file_menu(self, menubar):
605         file_menu = tk.Menu(menubar, tearoff=False)
606
607         file_menu.add_command(label="Open...", command=self.file_open_onclick)
608         file_menu.add_separator()
609         file_menu.add_command(label="Close", command=self.file_close_onclick)
610
611         return file_menu
612
613     def create_view_menu(self, menubar):
614         view_menu = tk.Menu(menubar, tearoff=False)
615
616         # self.view_menu.add_checkbutton(label="Hammer (raw)",
    variable=self.show_hammer_raw_plot,
617         #                                command=self.show_any_plot_onchanged)
618         # self.view_menu.add_checkbutton(label="Hammer (filtered)",
    variable=self.show_hammer_filtered_plot,
619         #                                command=self.show_any_plot_onchanged)
620         view_menu.add_checkbutton(label="Hammer",
    variable=self.show_hammer_raw_plot,
621                                   command=self.show_any_plot_onchanged)
622         view_menu.add_checkbutton(label="Hammer peaks",
    variable=self.show_hammer_peaks_plot,
623                                   command=self.show_any_plot_onchanged)
624         view_menu.add_checkbutton(label="EMG original",
    variable=self.show_emg_raw_plot,
625                                   command=self.show_any_plot_onchanged)
626         view_menu.add_checkbutton(label="EMG filtered",
    variable=self.show_emg_preprocessed_plot,
627                                   command=self.show_any_plot_onchanged)
628         # view_menu.add_checkbutton(label="EMG (filtered)",
    variable=self.show_emg_filtered_plot,
629         #                           command=self.show_any_plot_onchanged)
```

```
630         view_menu.add_checkbutton(label="EMG peaks",
        variable=self.show_emg_peaks_plot,
631                                   command=self.show_any_plot_onchanged)
632 
633         view_menu.add_separator()
634 
635         view_menu.add_command(label="Reset zoom",
        command=self.view_reset_zoom_onclick)
636 
637         return view_menu
638 
639     def create_emg_filter_menu(self, menubar):
640         emg_filter_menu = tk.Menu(menubar, tearoff=False)
641 
642         emg_filter_menu.add_radiobutton(label="No filter",
        variable=self.emg_filter_name,
643                                         value='-',
        command=self.emg_filter_name_onchanged)
644         # emg_filter_menuemg_filter_menu.add_radiobutton(label="Default filter
        (rolling RMS)", variable=self.emg_filter_name,
645         #                                 value='default',
        command=self.emg_filter_name_onchanged)
646 
647         emg_filter_menu.add_separator()
648 
649         emg_filter_menu.add_radiobutton(label="Butterworth low-pass",
        variable=self.emg_filter_name,
650                                         value='butter_lowpass_filtfilt',
        command=self.emg_filter_name_onchanged)
651         emg_filter_menu.add_radiobutton(label="Butterworth high-pass",
        variable=self.emg_filter_name,
652                                         value='butter_highpass_filtfilt',
        command=self.emg_filter_name_onchanged)
653         emg_filter_menu.add_radiobutton(label="Butterworth band-pass",
        variable=self.emg_filter_name,
654                                         value='butter_bandpass_filtfilt',
        command=self.emg_filter_name_onchanged)
```

```
655
656         # emg_filter_menu.add_radiobutton(label="Butterworth low-pass (causal) filter", variable=self.emg_filter_name,
657         #                                 value='butter_lowpass_lfilter', command=self.emg_filter_name_onchanged)
658         # emg_filter_menu.add_radiobutton(label="Chebyshev (type2) low-pass + filtfilt filter",
659         #                                 variable=self.emg_filter_name,
660         #                                 value='cheby2_lowpass_filtfilt', command=self.emg_filter_name_onchanged)
661
662         # emg_filter_menu.add_radiobutton(label="Notch + filtfilt filter", variable=self.emg_filter_name,
663         #                                 value='notch_filtfilt', command=self.emg_filter_name_onchanged)
664
665         # emg_filter_menu.add_radiobutton(label="Rolling RMS filter", variable=self.emg_filter_name,
666         #                                 value='rolling_rms', command=self.emg_filter_name_onchanged)
667         emg_filter_menu.add_radiobutton(label="SavitzkyGolay filter", variable=self.emg_filter_name,
668                                         value='savgol', command=self.emg_filter_name_onchanged)
669
670         # emg_filter_menu.add_radiobutton(label="Conventional EMG processing", variable=self.emg_filter_name,
671         #                                 value='conventional', command=self.emg_filter_name_onchanged)
672         # emg_filter_menu.add_radiobutton(label="pyemgpipeline EMG processing", variable=self.emg_filter_name,
673         #                                 value='pyemgpipeline', command=self.emg_filter_name_onchanged)
674
675         # emg_filter_menu.add_separator()
676         #
677         # emg_filter_menu.add_radiobutton(label="Gradient",
```

```
variable=self.emg_filter_name,
678         #                                   value='gradient',
command=self.emg_filter_name_onchanged)
679
680         return emg_filter_menu
681
682     # def create_emg_preprocess_menu(self, menubar):
683     #     emg_preprocess_menu = tk.Menu(menubar, tearoff=False)
684     #
685     #     emg_preprocess_menu.add_command(label="Apply current filter",
686     #
command=self.emg_preprocess_apply_current_filter_onclick)
687     #
688     #     emg_preprocess_menu.add_command(label="Reset to raw signal",
command=self.emg_preprocess_reset_onclick)
689     #
690     #     return emg_preprocess_menu
691
692     def create_latency_menu(self, menubar):
693         latency_menu = tk.Menu(menubar, tearoff=False)
694
695         latency_menu.add_radiobutton(label="Local maxima",
variable=self.latency_postprocess_name,
696                                      value='max',
command=self.latency_postprocess_name_onchanged)
697         latency_menu.add_radiobutton(label="Local minima",
variable=self.latency_postprocess_name,
698                                      value='min',
command=self.latency_postprocess_name_onchanged)
699         latency_menu.add_radiobutton(label="Local maxima & minima",
variable=self.latency_postprocess_name,
700                                      value='max+min',
command=self.latency_postprocess_name_onchanged)
701         latency_menu.add_radiobutton(label="Derivatives maxima",
variable=self.latency_postprocess_name,
702                                      value='gradient',
command=self.latency_postprocess_name_onchanged)
```

```
703
704             latency_menu.add_separator()
705
706             latency_menu.add_command(label="Show statistics",
        command=self.latency_show_stats_onclick)
707
708             return latency_menu
709
710         def create_canvas(self, parent):
711             canvas = FigureCanvasTkAgg(self.lca_plot.figure, master=parent)
712             canvas.mpl_connect('scroll_event', self.canvas_onmousescroll)
713
714             return canvas
715
716         def create_toolbar(self, canvas, parent):
717             toolbar = NavigationToolbar2Tk(canvas, parent)
718             toolbar.update()
719
720             return toolbar
721
722         def run(self):
723             self.root_window.mainloop()
724
725         def file_open_onclick(self):
726             file_name = LcaFileUtils.ask_open_txt_file_name()
727             lca_data = LcaData(file_name)
728             lca_plot_window = LcaPlotWindow(lca_data)
729             lca_plot_window.run()
730
731         def file_close_onclick(self):
732             self.root_window.destroy()
733
734         def show_any_plot_onchanged(self):
735             self.refresh_plot()
736
737         def canvas_onmousescroll(self, event):
738             # get current x and y limits
```

```
739         xlim = self.lca_plot.emg_axes.get_xlim()
740         ylim = self.lca_plot.emg_axes.get_ylim()
741
742         event_key = event.key or ''
743
744         if event_key.find('alt') >= 0:
745             # pan
746             if event_key.find('shift') >= 0:
747                 # vertical panning
748                 x_pan_factor = 0
749                 y_pan_factor = 0.1
750             else:
751                 # horizontal panning
752                 x_pan_factor = 0.1
753                 y_pan_factor = 0
754
755             if event_key.find('control') >= 0:
756                 # fast horizontal panning
757                 x_pan_factor *= 5
758                 y_pan_factor *= 5
759
760             if event.button == 'up':
761                 x_pan = -x_pan_factor * (xlim[1] - xlim[0])
762                 y_pan = -y_pan_factor * (ylim[1] - ylim[0])
763             else:
764                 x_pan = x_pan_factor * (xlim[1] - xlim[0])
765                 y_pan = y_pan_factor * (ylim[1] - ylim[0])
766
767             new_xlim = (xlim[0] + x_pan, xlim[1] + x_pan)
768             new_ylim = (ylim[0] + y_pan, ylim[1] + y_pan)
769
770         else:
771             # zoom
772             if event_key.find('control') >= 0:
773                 # full zoom
774                 x_scale_factor = 1.5
775                 y_scale_factor = 1.5
```

```
776             elif event_key.find('shift') >= 0:
777                 # vertical zoom
778                 x_scale_factor = 1.0
779                 y_scale_factor = 1.5
780             else:
781                 # horizontal zoom
782                 x_scale_factor = 1.5
783                 y_scale_factor = 1.0
784
785             if event.button == 'up':
786                 x_scale = 1 / x_scale_factor
787                 y_scale = 1 / y_scale_factor
788             else:
789                 x_scale = x_scale_factor
790                 y_scale = y_scale_factor
791
792             new_xlim = ([event.xdata - (event.xdata - xlim[0]) * x_scale,
793                          event.xdata + (xlim[1] - event.xdata) * x_scale])
794             new_ylim = ([event.ydata - (event.ydata - ylim[0]) * y_scale,
795                          event.ydata + (ylim[1] - event.ydata) * y_scale])
796
797         # set new limits
798
799         self.lca_plot.hammer_axes.set_xlim(new_xlim[0], new_xlim[1])
800         self.lca_plot.hammer_axes.set_ylim(new_ylim[0] * 10, new_ylim[1] * 10)
801
802         self.lca_plot.emg_axes.set_xlim(new_xlim[0], new_xlim[1])
803         self.lca_plot.emg_axes.set_ylim(new_ylim[0], new_ylim[1])
804
805         self.canvas.draw_idle()
806
807     def refresh_plot(self):
808         self.lca_plot.hammer_raw_plot.set_visible(self.show_hammer_raw_plot.get())
809         #
    self.lca_plot.hammer_filtered_plot.set_visible(self.show_hammer_filtered_plot.get())
810         self.lca_plot.hammer_peaks_plot.set_visible(self.show_hammer_peaks_plot.get())
811         self.lca_plot.hammer_axes.legend()
```

```
812
813         self . lca_plot .emg_raw_plot. set_visible ( self .show_emg_raw_plot.get())
814
        self . lca_plot .emg_preprocessed_plot. set_visible ( self .show_emg_preprocessed_plot.get() )
815         self . lca_plot . emg_filtered_plot . set_visible ( self . show_emg_filtered_plot . get () )
816         self . lca_plot .emg_peak_plot. set_visible ( self .show_emg_peaks_plot.get())
817         self . lca_plot .emg_axes.legend()
818
819         self .canvas. draw_idle ()
820
821     def view_reset_zoom_onclick( self ):
822         self . lca_plot .reset_zoom()
823         self .canvas. draw_idle ()
824
825     def emg_filter_name_onchanged(self):
826         self . lca_plot . set_emg_filter ( self .emg_filter_name. get () )
827         self .canvas. draw_idle ()
828
829     def emg_preprocess_reset_onclick ( self ):
830         self . lca_plot .reset_emg_preprocessed ()
831         self .canvas. draw_idle ()
832
833     def  emg_preprocess_apply_current_filter_onclick ( self ):
834         self . lca_plot . apply_current_emg_filter ()
835         self .emg_filter_name. set ('-')
836         self .canvas. draw_idle ()
837
838     def latency_postprocess_name_onchanged( self ):
839         self . lca_plot . set_latency_postprocess ( self . latency_postprocess_name . get () )
840         self .canvas. draw_idle ()
841
842     def  latency_show_stats_onclick ( self ):
843         xlim = self . lca_plot .emg_axes.get_xlim()
844          stats_text = self . lca_data . calc_stats_text (xlim)
845         lca_stats_window = LcaStatsWindow( stats_text )
846         lca_stats_window . run ()
847
```

```
848
849 class LcaStatsWindow:
850     def __init__(self, stats_text):
851         self.stats_text = stats_text
852
853         self.root_window = tk.Tk()
854
855         self.create_root_window(self.root_window)
856
857     def create_root_window(self, root_window):
858         root_window.title("EMG latency statistics")
859         root_window.iconbitmap(r"lca.ico")
860
861         menubar = self.create_menubar(root_window)
862         root_window.config(menu=menubar)
863
864         frame = tk.Frame(root_window)
865         frame.pack(fill=tk.BOTH, expand=True)
866
867         text = tk.Text(frame, font=('Courier', 10), wrap=tk.NONE)
868         text.insert(tk.END, self.stats_text)
869         text.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
870
871         scrollbar = tk.Scrollbar(frame, command=text.yview)
872         scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
873         text.config(yscrollcommand=scrollbar.set)
874
875     def create_menubar(self, parent):
876         menubar = tk.Menu(parent)
877
878         file_menu = self.create_file_menu(menubar)
879         menubar.add_cascade(label="File", menu=file_menu)
880
881         return menubar
882
883     def create_file_menu(self, menubar):
884         file_menu = tk.Menu(menubar, tearoff=False)
```

```
885
886             file_menu.add_command(label="Save as...", command=self.file_saveas_onclick)
887             file_menu.add_separator()
888             file_menu.add_command(label="Close", command=self.file_close_onclick)
889
890             return file_menu
891
892         def file_saveas_onclick(self):
893             file_name = LcaFileUtils.ask_save_txt_file_name()
894             if file_name:
895                 with open(file_name, "w") as file:
896                     file.write(self.stats_text)
897
898         def file_close_onclick(self):
899             self.root_window.destroy()
900
901         def run(self):
902             self.root_window.mainloop()
903
904
905 def main() -> int:
906     file_name = LcaFileUtils.ask_open_txt_file_name()
907     if file_name == '':
908         return 1
909     else:
910         lca_data = LcaData(file_name)
911         lca_plot_window = LcaPlotWindow(lca_data)
912         lca_plot_window.run()
913         return 0
914
915
916 if __name__ == '__main__':
917     sys.exit(main())
```