Chair of Applied Mechanics
TUM School of Engineering and Design
Technical University of Munich

Machine Learning-based High Level Control for Lower Limb Exoskeletons

Auf maschinellem Lernen basierende High Level Steuerung für Exoskelette der unteren Gliedmaßen

Scientific work for obtaining the academic degree
Master of Science (M.Sc.)
at the TUM School of Engineering and Design of the Technical University of Munich

Supervisor Prof. dr.ir. Daniel J. Rixen
Chair of Applied Mechanics

Advisor Arian Kist, M.Sc.
Chair of Applied Mechanics

Submitted by Pierre Quentin Ngandjui Tiako

Submitted on August 5, 2024 in Garching

Abstract

The development of exoskeletons for the rehabilitation and assistance of disabled people has necessitated the development of high-performance controllers to reproduce complex movements. The addition of Machine Learning has opened up the possibilities for higher-performance controllers. In the case of a lower limb exoskeleton, one of the primary objectives of the high-level controller is to ensure consistent gait. This means generating gait trajectories, or controlling gait at every instant. In this work, carried out in collaboration with the TUM DASH student initiative, the aim is to create a high-level controller which, using Machine Learning methods such as Deep Neural Networks and Gaussian Process Regression, will generate gait trajectories for the hip and knee. Trajectories here are the set of positions the joint must take to complete a gait. These trajectories are generated by generating isu key points of the trajectory before being interpolated. This trajectory generation makes it possible to obtain an average relative RMSE of less than 10% compared to the real amplitude of the trajectory.

Zusammenfassung

Die Entwicklung von Exoskeletten für die Rehabilitation und Unterstützung von Menschen mit Behinderungen erforderte die Entwicklung von leistungsfähigeren Controllern, um komplexe Bewegungen nachvollziehen zu können. Das Hinzufügen von Machine Learning hat die Möglichkeiten für leistungsfähigere Controller auf hohem Niveau eröffnet. Im Falle eines Lower Limb Exoskeleton besteht eines der Hauptziele des High-Level-Controllers darin, einen konsistenten Gang zu gewährleisten. Dies geschieht durch die Erzeugung von Gangtrajektorien oder die Kontrolle des Gaits zu jedem Zeitpunkt. In dieser Arbeit, die in Zusammenarbeit mit der Studenteninitiative TUM DASH entstanden ist, soll ein High-Level-Controller entwickelt werden, der mithilfe von Machine-Learning-Methoden wie Deep Neural Networks und Gaussian Process Regression Gangtrajektorien für Hüfte und Knie generiert. Trajektorien sind hier die Gesamtheit der Positionen, die das Gelenk einnehmen muss, um einen Gang zu vollenden. Die Generierung dieser Trajektorien erfolgt über die Generierung von isu-Schlüsselpunkten der Trajektorien, bevor sie interpoliert werden. Diese Trajektoriengenerierung ermöglicht es, einen durchschnittlichen relativen RMSE von weniger als 10% im Vergleich zur tatsächlichen Amplitude der Trajektorie zu erhalten.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

This project was born out of the TUM DASH student initiative. It consists of a group of students who want to build a lower limb exoskeleton. An exoskeleton is a wearable electromechanical structure adapted to the human body (or part of it). The aim of the TUM DASH student initiative is to take part in the Cybathlon. The Cybathlon is an international competition where athletes with severe physical disabilities use state-of-the-art assistive technologies, among them robotic prostheses, exoskeletons, and brain-computer interfaces. It will be organized by the Swiss Federal Institute of Technology, or ETH Zurich, to promote the development and use of AT that greatly enhances the quality of life of people with physical disabilities. The main objectives of the Cybathlon are:

- **Innovation and Development:** Encourage the development of assistive devices that can be of high quality and standard to assist the living of people with disabilities.
- **Awareness:** Increase public awareness about challenges persons with disabilities face and about the potential of assistive technologies.
- **Collaboration:** Foster collaboration between researchers, engineers, and users of assistive technologies to improve their design and functionality.

The exoskeleton can be used for a variety of purposes:

- *Human power augmentations* : e.g. minimize fatigue/injuries in industrial field [19]
- *Haptic interactions*: e.g. Force feedback in VR [6]
- *Rehabilitation*: Therapy or compensation of a non functional limb [30]

In the context of Cybathlon, the 3rd use is the one on which the project will focus. A lower limb exoskeleton must be able to reproduce/compensate for mechanically complex movements such as gait. To achieve this, a sophisticated planning and control framework needs to be developed.

1.2 Objective

As part of the project to create a lower limb exoskeleton by the TUM DASH student initiative, the aim is to establish a high-level controller for the exoskeleton. Due to the limitations of

the project environment, the possible inputs for this controller are the kinematic parameters of the exoskeleton (actuated joint position and velocity) and the biomechanical parameters of the rider (age, height, weight, etc.). As output, we need to obtain for each Degree of Freedoms (DoF), the trajectory to be accomplished for a complete cycle of a gait. More precisely, the trajectory is the set of positions to be taken by a joint to complete a gait cycle. Among the various possible methods [3], between methods involving the elaboration of the kinematic model and the part of the human body concerned, or the use of existing models to deduce dynamic parameters, the choice fell on artificial intelligence methods.

An artificial intelligence method can be used to adapt to the exoskeleton pilot, using his biomechanical parameters as input to the model, and a correlation can be demonstrated between these parameters and the trajectory profile [44, 47]. This controller will be able to adapt to the walking speed and biomechanical parameters of the pilot. The trajectories that will be generated are going to be constrained to a flat surface walk; therefore, although slope inclination also affects these trajectories, in this case, it is not put into consideration. The machine learning models that will be involved in this are going to be trained offline before being added to the exoskeleton system. Also, the memory requirements for the model to be used have to be put into consideration.

A corresponding database for training would be required for the use of machine learning models. Since this is so time-consuming, in the present case it shall be constituted from gait databases available on the Internet for research purposes. This can again be the foundation for a TUM DASH student initiative working on their own database, inspired by the data collected. Finally, involved DoFs, for which trajectories and speed profiles should be generated, are the knee flexion, hip flexion, and hip abduction.

1.3 Outline

Next sections will introduce the background knowledge that is required to fully understand the project, work done so far in this area, controller itself and its performance.

Chapter 2 focuses on knowledge to understand the project. It goes through the hierarchization and the categorization of controllers; the gait cycle and the controllers this latter inspires. It describes the different parts of the DASH exoskeleton and explains in more detail what is required in supervised machine learning.

Chapter 3 will explore the various works that have been done in the field of high-level machine learning controllers.

A more detailed description of the controller and its different components is given in chapter 4.

Chapter 5 deals with the performance of the said controller.

Chapter 6 concludes the project and shows some future directions.

Chapter 2

Background and Fundamental Theory

In this chapter, we will present the different knowledge required for the subject of Machine learning high level control for lower limb exoskeleton. After studying the control framework of an exoskeleton and its various components, the specific field of controllers for lower limb exoskeletons will be addressed, with a focus on gait and how it works. And after a look at the components of the DASH exoskeleton, we'll move on to the Machine Learning concepts that are essential for the future.

2.1 Exoskeleton Control Framework

In order for an exoskeleton to assist the pilot in executing complex movements, it is necessary to establish a sophisticated control system for this exoskeleton. In the following, the typical hierarchization of such control framework will be studied followed by the categorization of existing controllers.

2.1.1 Controller Hierarchization

In the design of lower limb exoskeletons, a hierarchical controller is essential to ensure smooth, efficient, and safe operation. The hierarchical controller is typically structured into three levels: high-level control, mid-level control, and low-level control, each responsible for different aspects of the exoskeleton's functionality [66].A visualization of these typical levels can be seen in fig. 2.1. The following sections provide an overview of these three control levels, highlighting their specific roles and interactions.

High Level Control

The high-level control, also known as the perception layer as seen in fig. 2.1, is responsible for estimating the user's intentions and recognizing the activity mode and context. In the context of a lower limb exoskeleton, the mode of activity can be whether the person is walking or running, or the speed of movement. For context, an example might be the inclination of the slope on which he's walking, or the presence of obstacles in the path. This layer usually involves sophisticated algorithms and machine learning techniques to accurately interpret the user's volitional intent based on sensory stimuli. By understanding the user's intentions, the high-level control can make informed decisions about the desired movements of the exoskeleton [66].

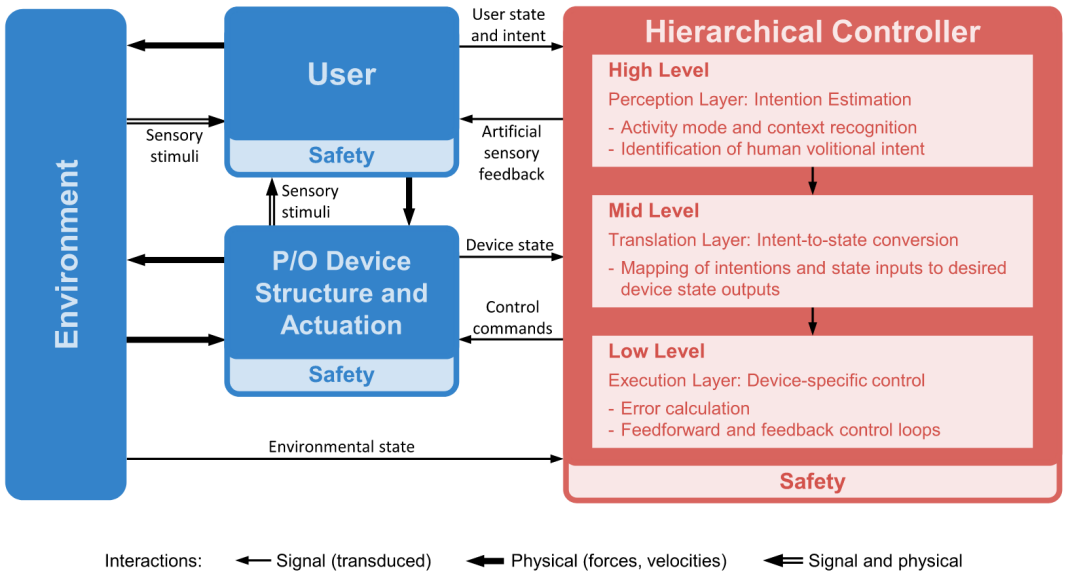


Figure 2.1: Generalized control framework for active lower limb prostheses and orthoses. Taken from [66]. The controller receives information, in the form of a transduced signal, from three entities: the user (exoskeleton pilot), the exoskeleton (P/O Device structure and actuation) and the environment. These signals reflect the state of these three entities. These states are affected by changes intrinsic to the entity (variations in the environment, for example), but also by interactions between the different entities. These interactions are essentially physical signals (exchanges of forces, for example), except for the user, who receives all information in the form of stimuli. The controller is organized into three levels, each with a specific role. The high level, whose objective is to use the states of the three preceding entities to establish and estimate the user's intention, set in the context of the state of the exoskeleton and the environment. The mid level translates the user's intention into the different states that the exoskeleton must take in order to achieve this intention. The low level ensures that the states defined by the mid level are reached as best as possible.

Controllers at this level can take several forms. Some allow, by taking electromyogram (EMG) signals from the muscles surrounding a joint as input, to output the force to be applied to it, as described in [30]. Others, using the kinematic state of the exoskeleton (joint speed and position) and the kinematic state of the user (leg angle and speed), can determine whether the user is running or walking in order to select the appropriate mid-level controller (classification 0 for walking and 1 for running) [66]. They can also determine the speed at which the user is moving and provide this information to the mid-level controller.

High-level control is crucial for adapting the exoskeleton's behavior to the user's needs and the environmental context, ensuring both safety and functionality. Recent advancements in machine learning have significantly enhanced the capability of high-level control systems to accurately predict and respond to user intentions [11, 15, 66].

Mid Level Control

The mid-level control, or translation layer in fig. 2.1, serves as the bridge between high-level intentions and low-level execution. It translates the estimated intentions from the high-level control into specific state commands that the exoskeleton's actuators can execute. This involves mapping the high-level goals to the desired device state outputs, ensuring that the exoskeleton's movements align with the user's intentions [66].

Mid-level controllers essentially take the information received from the high-level controller as input and output commands to drive the actuators. These outputs will therefore always be positions, speeds, or torques for the actuators. To revisit previous examples, the input might be the torque to apply to a joint or the user's walking speed. By their nature, it

often happens that the high-level and mid-level controllers are fused into one, as in the case of Eslamy et al [15], which takes the position and speed of the leg as input and deduces the necessary movements for the ankle actuators to follow.

Mid-level control plays a vital role in ensuring smooth and coordinated movement. It must effectively handle the conversion of abstract intentions into concrete actions, which requires a thorough understanding of both the user's biomechanics and the exoskeleton's mechanics [36, 41].

Low Level Control

The low-level control, or execution layer see fig. 2.1 is responsible for the direct control of the exoskeleton's actuators. This includes error calculation, feedforward, and feedback control loops to maintain precise and stable movement. Low-level control ensures that the commands from the mid-level control are executed accurately, compensating for any disturbances or deviations in real-time [66].

Effective low-level control is critical for the safety and performance of the exoskeleton. It must be capable of handling the high dynamics and variability in human movement, providing robust and reliable control under various conditions [30, 59].

2.1.2 Controller Categorization

The hierarchization of controllers allows for defining the different roles a controller can play within the control framework of an exoskeleton. It is also possible to differentiate controllers based on their constitution (their inputs and outputs, how they compute the outputs from the inputs, etc.). The hierarchization of controllers allows for defining the different roles a controller can play within the control framework of an exoskeleton. It is also possible to differentiate controllers based on their constitution (their inputs and outputs, how they compute the outputs from the inputs, etc.). This categorization of controllers is composed of two major groups : Model-based control system and physical parameters based control system [3]. Each of these categories or their sub-categories can be linked to a level in the controller hierarchy.

Model-based Control System

Model-based control systems uses a model of the human being to deduce the parameters required for the exoskeleton to function properly. Depending on the method used to obtain the model, there are two sub-categories: dynamic models and muscle models [3]. Due to the nature of this type of controller, the model can be used to recognize the exoskeleton user's intention and determine the variables needed to drive the actuators. As a result, they are often used as high or even mid-level controllers.

Dynamic Models Dynamic models are derived from modeling the human body using rigid elements, kinematic joints and the different forces exerted on them. Such a model can be obtained by several methods, ranging from the most theoretical to the most computational: mathematical method, system identification method and artificial intelligent method. The first, the most difficult to compute, requires the ability to theoretically model the entire human body (or part of the body concerned) using the equation of motion with the different joints as degree of freedom. The system identification method involves using data with existing mathematical methods/models to deduce dynamic parameters. These two methods are mainly used for BLEEX [33], the second being used to model a particular phase of the gait,

the swing phase fig. 2.5. During this phase, the leg is model as a 7 DoF serial link mechanism in the sagittal plane. The equation of motion for BLEEX during the swing phase is given by:

$$M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + P(\theta) = T + d \tag{2.1}$$

where:

- $M(\theta)$ is the inertia matrix,
- $C(\theta, \dot{\theta})$ is the centripetal and Coriolis matrix,
- $P(\theta)$ is the gravitational torque vector,
- T is the actuator torque vector,
- d represents the torque imposed by the pilot.

Then they used a system identification method to establish the inverse kinematics of the system and obtain the required torques. Finally, the artificial intelligence method, which is the focus of this study, will be presented in chapter 3.

Muscle Models Muscle models are specific in that, unlike dynamic models which only use kinematic and/or dynamic information from the system to model it, these use Electromyography signals to deduce the user's intention, or more precisely to estimate the force that the muscle whose signals are being measured wants to apply. A representation of how this type of model works can be found in fig. 2.4 taken from [30]. In fig. 2.4, the EMG signals are measured on the user and fed into a neural network for interpretation. From this interpretation, the neural network will deduce the user's activity mode and select the most appropriate mid-level controller. In the context of gait, this can determine whether the user is running or walking and select the mid-level controller accordingly.

There is a difference in the way the model is parameterized. A muscle model can be parametric, i.e. it uses a representation of the muscle (Hill-based [53] for example) and dynamic joints to deduce an estimate of the force applied, with EMG signals as input. It can also be non-parametric, in which Input/Output data is used with which a model is trained. A comparison between the performance of these two types can be found here [53], with a better prediction in the case of the use of a neural network. But more recent studies have tended to use the non-parametric approach with neuro fuzzy logic, i.e. neural networks used in the interpretation of fuzzy logic. Fuzzy logic is a variety of multivalued logic derived from fuzzy set theory and aims to deal with approximate rather than precise reasoning. Whereas the variables in the classic binary logic should be either true or false, in fuzzy logic, they can have truth values ranging from 0 to 1 [60]. The key concept in fuzzy logic is the membership function, which defines the degree of truth as an extension of valuation. In this respect, this is much more flexible and closer to the way human reasoning and decision making go, especially in complex systems wherein binary logic goes lame. The first use of this method was to improve the recognition of intention by EMG signals [34]. The memberships functions in fig. 2.2 are used to represent for the elbow and shoulder at which angle they are considered flexed, intermediate or extended, and to what extend an angle belongs to one of those categories. There are other membership function but for muscles to know if there are flexed or extended. This method has become prevalent in this type of control, notably adding physical parameters based control systems like PD controllers [42, 49] which will be explained below.

Other methods are also possible in the case of non parametric methods, as for example in [16], where a nonlinear auto regressive model with exogenous output was used with

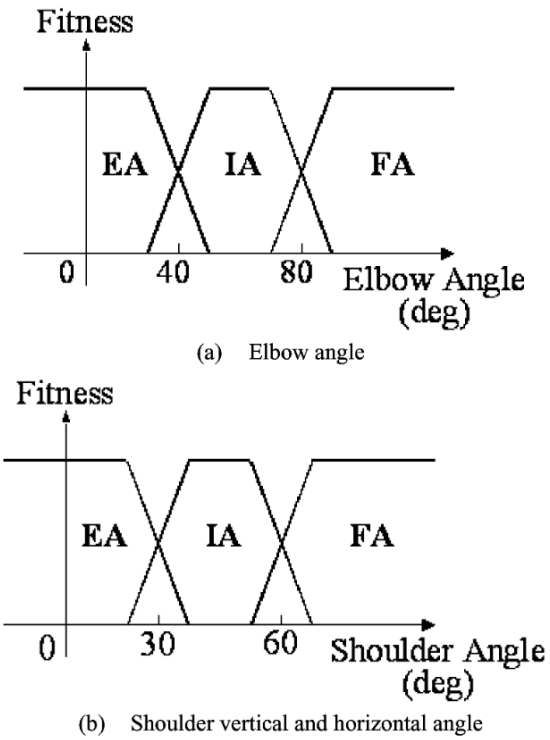


Figure 2.2: Fuzzy Logic in the context of Kiguchi et al work [34] It reprsents the menberships function used in their work. FA: flexed angle, IA: intermediate angle, and EA: extended Angle. The shoulder and elbow are consider flexed or extended at different angle and at some angles there can be flexed and intermediate at the same time. Those membership function represents at which moment angle the joint is flexed, intermediate or extended and to what entend the angle belongs to one of the categories.

satisfactory results. A nonlinear autoregressive model exogenous (NARX) input is one of the techniques of time series modeling applied in the prediction of a system's future values based on past values and some other external inputs. This model is especially useful for capturing complex, nonlinear relationships in data. Using a delayed measurements of EMG signals and the previous ankle estimates, they are able to continously predict the ankle angle over time.

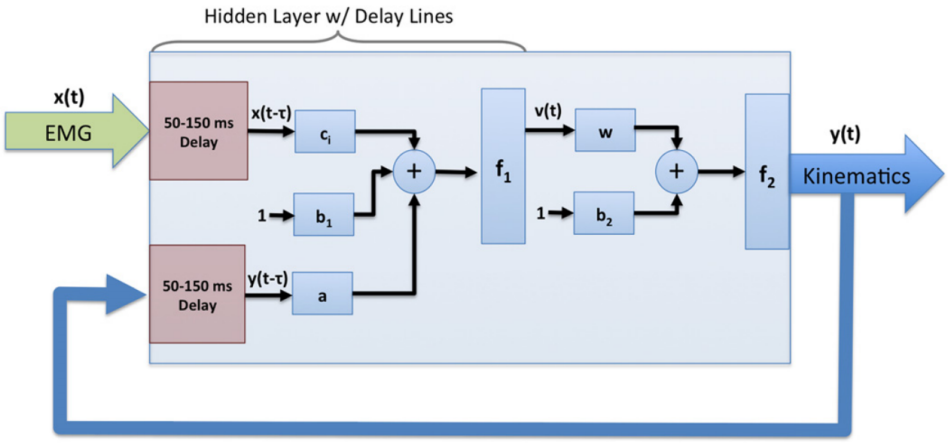


Figure 2.3: Non Linear Autoregressive model with exogenous output used by [16]. Windowed EMG activity and previous estimates of ankle angle were weighted and fed via tapped delay lines to a hidden layer comprised of nonlinear units. Outputs from the hidden layer were weighted and linearly combined to provide a continuous estimate of ankle angle over time.

Also, without going as far as force estimation, the use of EMG signals for control has also been used solely for intent recognition, i.e. in cases of high level controller only. This is the case with [71], which uses EMG signals and mechanical information from various sensors to deduce the user's intention. The objective is to better understand the contribution of EMG signals compare to various mechanical sensors.

Studies such as [41], seek to compare the use of EMG signals in control compared to just exclusively mechanical parameters. In this case, the use of EMG signals could be better in the control of exoskeletons, particularly in the context of rehabilitation, but more in-depth studies are needed.

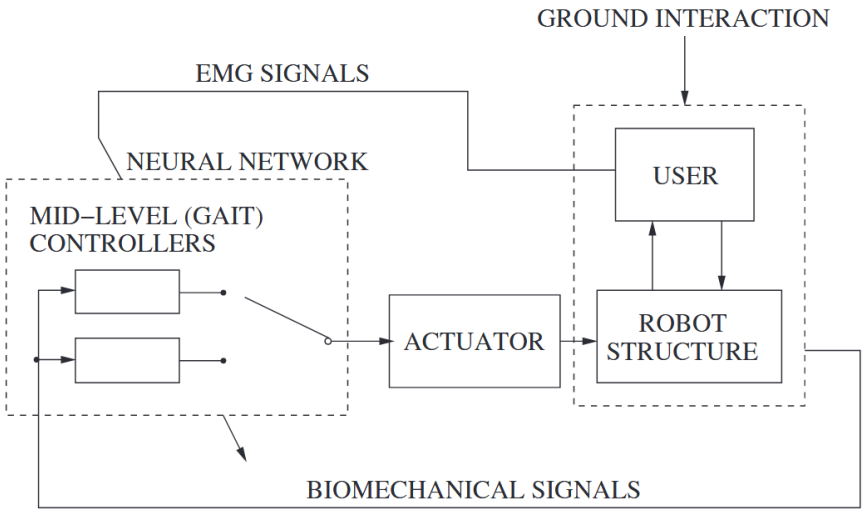


Figure 2.4: Control system diagram using EMG signals for intent recognition. Taken from [30]. An architectural version of control using EMG signals where these are used in parallel with biomechanical signals from the user/device assembly for intention recognition. The EMG signals measured from the user are given to a NN that is able to recognize the ability mode of the user and chose the right mid level controller for this activity mode. They will the pilot the actuators in an adapted way given the context and the biomechanicals signals they receive from the robot structure.

Physical Parameters-based Control System

Physical parameters-based control refers to a control strategy that utilizes measurable physical parameters of a system to regulate its behavior. These parameters could include quantities such as force, torque, pressure, temperature, displacement, velocity, and acceleration. Physical parameters based control system is the basis of any control system, particularly as all the controllers in this category are the closest to actuators, i.e. low level controllers, except in certain cases. This type of controller can be found in all conventional mechatronic systems. They can be classified into two types of controller: for position and for force/torque. As far as position controllers are concerned, the best-known are PID controllers. In the case of force/torque controllers, it is possible to use them as mid level controllers when the aim is to control the force of interaction between the human and the exoskeleton [3]. This type of controller applied in this context are in the form of impedance controllers [67] or admittance controllers [2]. Impedance controllers tend to use the force as controlled parameter to pilot the position and admittance controller use the position as controlled parameter to pilot the force. This makes it possible to aid rehabilitation by using these interaction forces to correct walking errors, as in the case of [2, 67].

2.2 Gait-based Control

The most common purpose for a lower limb exoskeleton is the replication of human gait. Lower limb exoskeleton controllers must be able to reproduce and support the rider in the execution of the gait. In the following, we'll describe the gait process in more detail, and how controllers for this process are generally designed.

2.2.1 Human Gait Description

Gait, the manner of walking or moving on foot, is a complex and dynamic process involving the coordinated action of muscles, joints, and neural mechanisms [61], which can be observed on fig. 2.5. Understanding the gait process is essential for fields such as biomechanics, rehabilitation, and robotics in the context of lower limb devices. This chapter provides a detailed description of the gait process, its different phases, and their key features. The terms of fig. 2.5 will be explained in the following paragraphs.

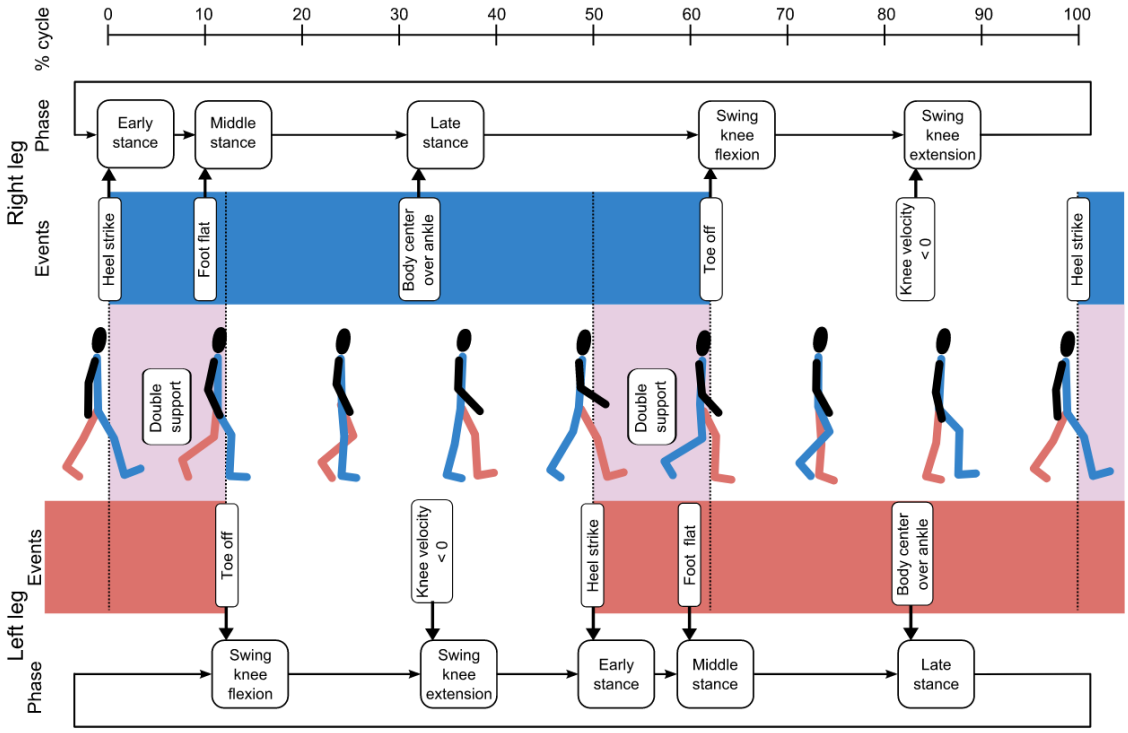


Figure 2.5: Finite-state decomposition of level human gait. Taken from [66]. Steady-state locomotion can be represented as a periodic sequence of states (or phases), where the transitions between the states are triggered by events within the gait cycle. The choice of the number of states and the type of events used are somewhat arbitrary, and will depend on what information is available from the sensors and which joint the device is to actuate. In this example for the knee joints, stance has been divided into three states, with early and middle stance initiated by ground contact events at the heel and toe of the foot, for example determined using pressure sensitive insoles.

Phases of the gait

The gait cycle can be divided into two primary phases: the stance phase and the swing phase [61]. Each phase encompasses specific movements and functions that contribute to efficient locomotion.

Stance Phase The stance phase accounts for approximately 60% of the gait cycle [61]. It begins when the heel contacts the ground (heel strike) and ends when the toes leave the ground (toe-off). This phase can be further subdivided into the following periods [40]:

- **Initial Contact**: The moment when the foot first touches the ground. It is characterized by the heel strike, where the foot transitions from the swing phase to the stance phase.
- **Loading Response**: Following initial contact, the foot absorbs the body's weight, and the entire foot comes into contact with the ground. This period is crucial for shock absorption and stability.
- **Middle stance**: This is when the body progresses over the stationary foot. The body's center of gravity is directly over the supporting limb, providing balance and support.
- **Late Stance**: The body continues to move forward, with the heel rising from the ground. This phase ends when the opposite foot strikes the ground.
- **Pre-Swing**: Also known as the push-off phase, this period involves the transfer of weight to the opposite limb, and the foot prepares to leave the ground.

Swing Phase The swing phase comprises the remaining 40% of the gait cycle [61]. It begins when the foot leaves the ground and ends when it contacts the ground again. This phase can be divided into [40]:

- **Initial Swing**: The foot is lifted off the ground, and the leg accelerates forward. This phase is critical for initiating limb advancement, correspond infig. 2.5 to the swing knee flexion.
- **Midswing**: The leg continues to advance as the knee begins to extend, and the foot moves forward past the opposite stance limb, correspond infig. 2.5 to the swing knee extension.
- **Terminal Swing**: The final period of the swing phase, where the knee fully extends, and the foot prepares for initial contact with the ground.

2.2.2 Gait-based controllers

The aim of gait-based controllers is to ensure stable gait via the exoskeleton. To achieve this, exoskeleton actuators are connected to the various joints (hip, knee, ankle) so that they can be controlled. For each joint, there are three possible DoFs for the hip, one for the knee and three for the ankle [10] as represented in fig. 2.6:

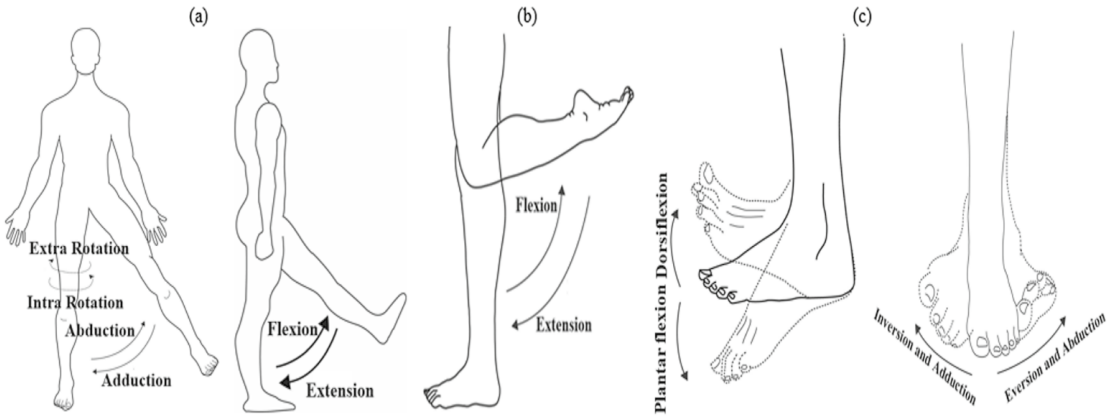


Figure 2.6: Different DoFs for the lower limb joints. Taken from [32]. Different joint movement of the lower limb (a) hip abduction/adduction (a/a), intra/extra rotation (i/e) and flexion/extension (f/e), (b) knee flexion/extension (f/e), (c) ankle dorsi/planter flexion (d/p), abduction/adduction (a/a), and ankle inversion/eversion (i/e). The definition of those DoFs will be given in the following paragraph.

- **Flexion / Extension** refers to the movement that decreases / increases the angle between two body parts. In the context of human joints, flexion typically involves bending a limb at a joint, bringing the two parts closer / further together.

- **Abduction** is the movement of a limb or body part away from the midline of the body. In other words, it is the action of drawing away from the central axis of the body or limb. **Adduction** is the movement of a limb or other part toward the midline of the body or towards another part. It is the opposite of abduction.

- **Internal rotation**, also known as medial rotation, involves the rotation of a limb or body part towards the midline of the body. **External Rotation**, also known as lateral rotation, involves the rotation of a limb or body part away from the midline of the body.

- **Dorsiflexion** is the action that reduces the angle between the dorsum (top) of the foot and the leg, drawing the toes toward the shin. It is also known as raising the front of the foot upward. **Plantar Flexion** is the opposite movement, which is the increase in the angle between the dorsum of the foot and the leg. It is a movement pointing the toes downward, away from the shin, such as pressing on a gas pedal.

- **Inversion** refers to the movement of the sole (flexor surface of the foot) towards the median plane, and it means the sole turns inwards. The motion decreases the angle between the medial aspect of the foot and the midline of the body. **Eversion** is the turning of the sole away from the median plane, where it is oriented outward. The movement increases the angle between the medial side of the foot and the midline of the body.

But this does not imply that all these DOFs are actively controlled. In the paper [10], lower-limb exoskeletons such as the one at the Massachusetts Institute of Technology are almost completely passive, meaning that there are no actuators to drive the DoF, but the device structure allows movement according to the Dof, and even in most active exoskeletons, only two out of three DOFs for the hip and ankle are considered. And there are even models that are only actively control one joint. Among the various parameters to be taken into account when designing a gait based controller, the number of DOFs or actuators to be controlled must be considered.

With the aim of reproducing the gait cycle for use in high level control, fig. 2.5 shows 2 main ways of doing this:

- A first consideration for such a controller would be to consider the different phases of gait separately. It is possible to define different mid level controllers specialized in piloting the exoskeleton in a specific phase (stance or swing) or define multiple mid level controllers depending on the activity mode, one for walking gait, one for running gait for example. We would then have a high level controller which would allow us to determine which phase of gait the individual or what the activity mode is in, and then choose the appropriate mid level controller, knowing that each of them corresponds to a specific phase. A representation of this can be seen in figure fig. 2.7 from [30]. In the case of fig. 2.7 taking the kinematic state of the system (Robot structure + User), be able to recognize the activity mode (walking, running, etc) and then choose the mid level controller adapted to it.

- The second case is to consider the gait cycle in its whole and to make a temporal discretization of it. More specifically, for each of the joints (hip, knee, ankle), the aim is to be able to represent the trajectory of this joint (angle, speed or torque) as a function of the moment in the gait cycle. In the simplest cases, such as when a single joint is being controlled, it is possible, as shown in the figure fig. 2.8 from [12], to predict at each moment of the gait cycle what the next position or speed of the joint should be. In one version of this theory, there is, for example, the article [13], an active exoskeleton for the knee, where there is one controller (no multiple mid level controllers) that continuously pilot the knee no matter the gait phase or the activity mode of the user.

The prevalence of being able to develop controllers that consider gait as a continuous process over time and not by dividing it into phases has been noted in recent articles. As far as one-joint exoskeletons are concerned, this method has been used extensively recently, including the various examples cited recently in [9], which establishes a relationship between the kinematic and dynamic parameters of the hip and knee joints and that of the ankle being controlled. The discovery of this potential correlation, or even synergy, between the angles and speeds of the various joints from the shoulder to the knee, with the right side influencing the left side and vice versa, is taken even further in the article [11]. This makes it possible to use information other than that from the lower limb to drive a lower limb exoskeleton. As a result, there has been a resurgence of work that provides a model that can continuously reconstruct the gait profile of an individual's different joints, as in the case of [14], who uses and improves on the technique used by [70]. They use thigh- and leg-related angles, velocities and moments as input, and legacy predictions of joint angles (hip, knee, ankle) to continuously predict the angles of different joints using neural network wavelets.

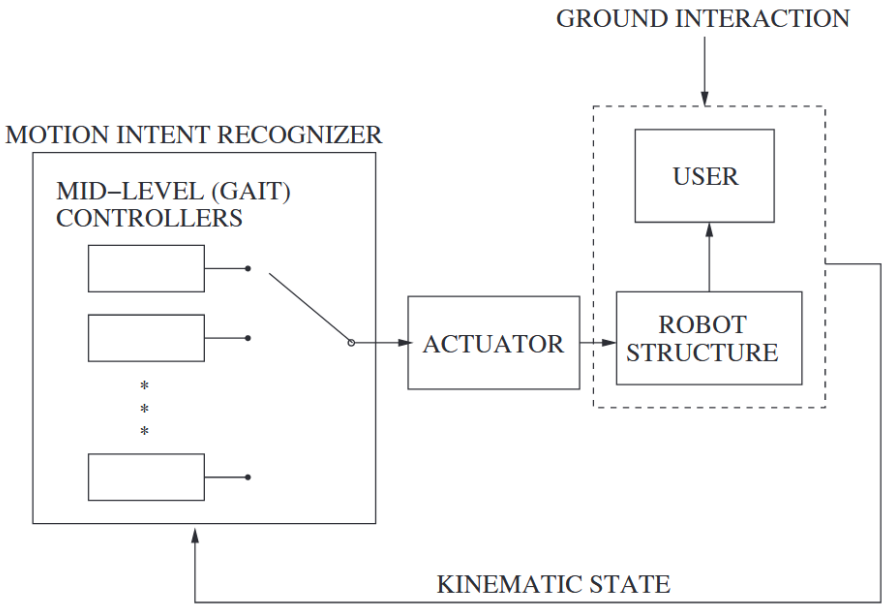


Figure 2.7: Diagram of a controller with a gait cycle divided into phases. Taken from [30]. In this control architecture, the kinematic state of the user and exoskeleton is used to predict the user's intention. More specifically, in the case of gait-based control, the aim here is to determine which phase of gait the user is in in order to assign control to the mid-level controller best suited to this phase..

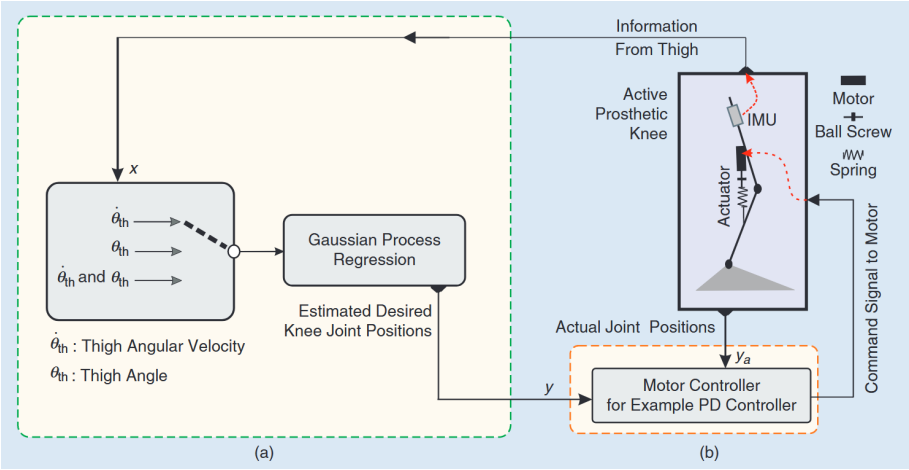


Figure 2.8: Control of an active knee exoskeleton. Taken from [12]. An example of machine learning based control for a one-joint exoskeleton. The model used is Gaussian Process Regression, with the input being the angular position, angular velocity or a combination of the two, and the output being the next desired position for the knee.

2.3 DASH Exoskeleton Hardware and Software

The DASH exoskeleton, code-named "ASH," is a highly complex robotic system for enhancing human mobility. This section describes its hardware and software components.

2.3.1 Sensors

- **Foot sole based on Velostat architecture:** Detects pressure and distribution across the foot sole, crucial for balance and gait analysis.
- **Temperature sensors:** Monitor the temperature of various components, ensuring they are within safe operating limits and preventing overheating.
- **Computer vision (planned for future integration):** Enhances the exoskeleton's capability to interpret and react to the environment, improving navigation and interaction.
- **Battery level monitoring:** Ensures the user is aware of the power status, preventing unexpected shutdowns and maintaining reliable operation.

2.3.2 Power System

- **60A fuse boxes:** Provide overcurrent protection, preventing damage to electrical components and ensuring user safety.
- **Power distribution system:** Manages and distributes electrical power efficiently across all components of the exoskeleton.
- **Active and passive cooling systems:** Maintain optimal operating temperatures for electronics and motors, ensuring performance and longevity.
- **Power distribution at actuator level:** Ensures that actuators receive the necessary power for precise and responsive control.

2.3.3 Control System

- **Low-level actuator specific control:** Provides fine-tuned control over individual actuators, crucial for precise movement and responsiveness.
- **Mid-level trajectory following and multi-dimensional impedance control:** Ensures smooth and adaptive movement by following predefined paths and adjusting to dynamic forces.
- **High-level live trajectory generation and adaptation:** Enables real-time adjustment to movement trajectories based on sensor input and environmental changes.
- **Joint specific controllers:** Manage each joint's movements independently, allowing for complex and coordinated actions.
- **Real-time operating system:** Ensures timely and reliable execution of control algorithms, critical for maintaining synchronization and performance.

2.3.4 Mechanical Structure

- **Frame backpack:** Provides a supportive structure to distribute the weight of the exoskeleton, improving comfort and usability.

- **Knee joints with variable torque on angle output relations (non-symmetric Jacobian):** Allows for dynamic and responsive knee movement, enhancing walking and other leg movements.
- **Passively actuated ankle joints:** Simplify the design and reduce power consumption while still supporting necessary ankle movements.
- **Hip flexion and extension:** Facilitates essential hip movements, contributing to natural and effective locomotion.
- **Hip action and abduction/adduction:** Enhances lateral and rotational hip movements, supporting a wider range of activities.
- **Carbon fiber frame attachments and protection:** Provide strength and durability while minimizing weight, ensuring protection and longevity.

2.3.5 Pilot Interaction and Support

- **Pilot training related handles and equipment:** Facilitate training and familiarization with the exoskeleton, ensuring users can operate it effectively.
- **Pilot interfaces and supporting mechanisms:** Improve usability and comfort, enabling the pilot to interact with and control the exoskeleton seamlessly.
- **Ergonomic handles and push-to-go buttons on the handles:** Enhance user comfort and control, making the exoskeleton more intuitive to use.

2.3.6 Embedded Systems

- **Controlling PCBs (managing the interfaces on every joint):** Centralize control and communication for each joint, ensuring coordinated and precise movements.
- **Main computer and cabling:** Serve as the central processing unit, managing data and control signals throughout the exoskeleton.
- **Doubly redundant encoder schemes (including absolute encoders):** Ensure reliable position sensing and control accuracy, even if one encoder fails.
- **Protection measures:** Safeguard the electronics and user from electrical faults and other hazards.
- **Exoskeleton-wide I2C and SPI bus interfaces:** Enable efficient and reliable communication between components, ensuring synchronized operation.

2.3.7 Human-Machine Interface (HMI)

- **Joystick and mode selection:** Provide intuitive control over the exoskeleton's movements and modes, enhancing user experience.
- **Status display and Finite State Machine (FSM) state selection:** Allow the user to monitor the exoskeleton's status and select operational states, improving control and situational awareness.

- **Emergency buttons:** Offer immediate stop functionality, ensuring user safety in case of unexpected situations or malfunctions.

2.4 Supervised Machine Learning

Supervised Machine Learning is a fundamental branch of machine learning where the goal is to learn a mapping from input features to output labels based on a set of example input-output pairs. This learning process is termed "supervised" because it operates under the supervision of labeled training data, where the correct output is provided for each input [27]. This enables the algorithm to learn the relationship between inputs and outputs, and subsequently make predictions on new, unseen data.

In what follows, we'll explore the importance of data standardization for supervised Machine Learning, and look at the two models in this category that will be used next, Gaussian Process Regression and Deep Neural Networks.

2.4.1 Standardization of Data

In machine learning, standardization of data is a crucial preprocessing step that involves transforming the features of the dataset to have a mean of zero and a standard deviation of one [28]. This transformation is essential for ensuring that each feature contributes equally to the model's performance and that the model can learn efficiently from the data.

Standardization, also known as Z-score normalization, transforms the data using the following formula:

$$z = \frac{x - \mu}{\sigma} \tag{2.2}$$

where x represents the original data point, μ is the mean of the training data, and σ is the standard deviation of the training data [28]. By applying this transformation, the resulting standardized data will have a mean of zero and a standard deviation of one. This process ensures that the data is centered around zero and that the variance is uniform across all features [28].

Many machine learning algorithms, especially those that rely on gradient-based optimization methods (such as gradient descent), perform better when the data is standardized. When features are on different scales, the optimization algorithm can become biased towards the features with larger scales, leading to inefficient learning and slow convergence. Standardization mitigates this issue by ensuring that all features are on a comparable scale [28].

Models such as support vector machines (SVMs), k-nearest neighbors (k-NN), and principal component analysis (PCA) are sensitive to the scale of the data. Standardizing the features can significantly improve the performance of these models by preventing any single feature from disproportionately influencing the results. This leads to more accurate and reliable predictions [21].

2.4.2 Gaussian Process Regression

Gaussian Process Regression (GPR) is a non-parametric, Bayesian approach to regression that is particularly useful for modeling uncertain data and the non linear relationships between

them [51]. Unlike traditional regression methods that assume a specific form for the underlying function, GPR defines a distribution over possible functions and makes predictions based on the observed data. GPR offers a flexible and powerful approach to modeling complex data by defining a distribution over functions. The choice of kernel plays a critical role in determining the properties of the functions that the Gaussian Process (GP) can model [52]. n the context of a GP, the kernel determines how points in the input space are correlated with each other. This correlation is critical because it allows the GP to make predictions about unknown data points based on the known data points. By selecting an appropriate kernel, we can incorporate prior knowledge about the function and capture various patterns in the data.

Overview of Gaussian Process Regression

A Gaussian Process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution [52]. Formally, a GP is defined as a set of random variables $\{f(x)\}_{x\in\mathcal{X}}$, such that any finite subset $(f(x_1), f(x_2), \ldots, f(x_n))$ has a multivariate Gaussian distribution:

$$(f(x_1), f(x_2), \ldots, f(x_n)) \sim \mathcal{N}(\mu, K) \tag{2.3}$$

where $\mu = [\mu(x_1), \mu(x_2), \ldots, \mu(x_n)]^T$ is the mean vector and K is the covariance matrix with elements $K_{ij} = k(x_i, x_j)$.

The GP is fully specified by its mean function $m(x)$ and covariance function (or kernel) $k(x, x')$ [52]:

$$m(x) = \mathbb{E}[f(x)] \tag{2.4}$$

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \tag{2.5}$$

Given a set of training data points $\{(x_i, y_i)\}_{i=1}^n$, where y_i are the observed values and x_i are the input locations, we assume that the observations are generated from the underlying function with some added Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$:

$$y_i = f(x_i) + \epsilon \tag{2.6}$$

Prior and Posterior in Gaussian Processes

Before observing any data, we place a prior on the function values f, assuming $f \sim \mathcal{GP}(m(x), k(x, x'))$. The term "prior" is used because it captures our assumptions or knowledge about the function's behavior prior to seeing any evidence. This prior captures our initial beliefs about the function's behavior, defined by:

- Mean Function ($m(x)$): Describes the expected value of the function at each point in the input space. Often set to zero if no specific prior knowledge about the mean is available.
- Covariance Function (Kernel, $k(x, x')$): Describes how function values at different points in the input space are correlated. The kernel determines properties such as

smoothness and periodicity of the functions. The kernel is composed of a constant part and a variable part. The constant part of the kernel function is responsible for capturing the global properties of the function, such as its overall variance or bias. The variable part of the kernel captures how the function values vary with changes in the input. It usually depends on the distance or relationship between input points and is responsible for modeling local properties like smoothness, periodicity, or more complex interactions.

After observing data, we update our beliefs to obtain the posterior distribution over the function values at new input locations $X_* = \{x_*^{(i)}\}_{i=1}^m$. The posterior represents updated beliefs about the function after incorporating the observed data. It is called "posterior" because it comes after the observation. The joint distribution of the observed target values $\mathbf{y}$ and the function values at the new inputs $\mathbf{f}_*$ is given by:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m} \\ \mathbf{m}_* \end{bmatrix}, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix} \right) \tag{2.7}$$

where $K(X,X)$ is the covariance matrix evaluated at the training inputs, $K(X,X_*)$ is the covariance matrix between the training inputs and the test inputs, and $K(X_*,X_*)$ is the covariance matrix at the test inputs.

The posterior distribution of the function values at the new inputs $\mathbf{f}_*$ given the observed data is also Gaussian:

$$\mathbf{f}_* \mid X, \mathbf{y}, X_* \sim \mathcal{N}(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)) \tag{2.8}$$

where the posterior mean $\bar{\mathbf{f}}_*$ and covariance $\text{cov}(\mathbf{f}_*)$ are given by:

$$\bar{\mathbf{f}}_* = \mathbf{m}_* + K(X_*,X)[K(X,X) + \sigma_n^2 I]^{-1}(\mathbf{y} - \mathbf{m}) \tag{2.9}$$

$$\text{cov}(\mathbf{f}_*) = K(X_*,X_*) - K(X_*,X)[K(X,X) + \sigma_n^2 I]^{-1} K(X,X_*) \tag{2.10}$$

This posterior distribution allows us to make predictions about the function values at new input locations, incorporating both the mean prediction and the uncertainty.

Kernels in Gaussian Process Regression

The choice of kernel (or covariance function) is crucial in GPR, as it encodes our assumptions about the function we wish to learn [56]. Different kernels can capture different types of patterns in the data. In **??** are some commonly used kernels and their specificities:

Table 2.1: Comparison of Different Kernels for Gaussian Process Regression

Kernel	Definition	Specificities
Squared Exponential (RBF) [52]	$k(x,x') = \sigma_f^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$ **Elements:** • σ_f^2: Signal variance. • ℓ: Length scale.	• Captures smooth and continuous functions. • The length scale ℓ determines the extent of correlation between points; smaller ℓ leads to more rapidly varying functions.
Matérn [56]	$k(x,x') = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}\lvert x-x'\rvert}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}\lvert x-x'\rvert}{\ell}\right)$ **Elements:** • ν: Controls the smoothness of the function. • ℓ: Length scale. • K_ν: Modified Bessel function. • $\Gamma(\nu)$: Gamma function.	• Can model functions with varying degrees of smoothness. • When $\nu = 0.5$, it becomes equivalent to the absolute exponential kernel; higher ν leads to smoother functions.
Rational Quadratic [52]	$k(x,x') = \sigma_f^2\left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$ **Elements:** • σ_f^2: Signal variance. • ℓ: Length scale. • α: Controls the relative weighting of large and small-scale variations.	• Can model functions with varying length scales. • When $\alpha \to \infty$, it approaches the RBF kernel.
ExpSineSquared [43]	$k(x,x') = \sigma_f^2 \exp\left(-\frac{2\sin^2\left(\frac{\pi\lvert x-x'\rvert}{p}\right)}{\ell^2}\right)$ **Elements:** • σ_f^2: Signal variance. • ℓ: Length scale. • p: Period.	• Ideal for capturing smooth periodic patterns. • Can handle multiple periodic components by summing multiple ExpSineSquared kernels.
DotProduct [51]	$k(x,x') = \sigma_0^2 + x \cdot x'$ **Elements:** • σ_0^2: Constant term.	• Suitable for linear regression tasks. • Can be used in combination with other kernels to model more complex functions.

2.4.3 Deep Neural Networks

A Deep Neural Network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. These layers, also known as hidden layers, enable the network to non-linear relationships in the data [27]. Each layer in a DNN consists of neurons, which are computational units that process the input and pass the transformed data to the next layer.

The architecture of a DNN typically includes an input layer, several hidden layers, and an output layer as in fig. 2.9. The input layer receives raw data, such as images, text, or numerical values, which are then processed through the hidden layers. Each hidden layer applies a set of weights and an activation function to the inputs from the previous layer to produce an output. This output serves as the input for the next layer. The final output layer produces the prediction or classification based on the cumulative processing of all previous layers [38].

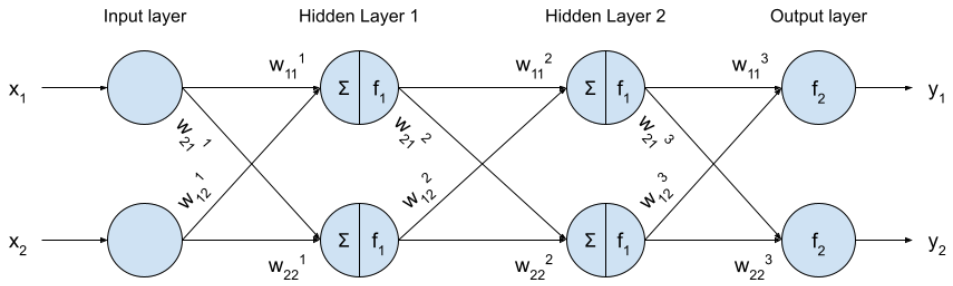


Figure 2.9: Schematic diagram of a DNN with two hidden layers with activation functions and weights. f_1 and f_2 are the activation functions use in this case and w_{ij}^k with i,j = 1,2 and k = 1,2,3 are the weights of the DNN. For example, the outputs for the first layers are: $x_1^1 = f_1(w_{11}^1 x_1 + w_{12}^1 x_2 + b_1^1)$ and $x_2^1 = f_1(w_{21}^1 x_1 + w_{22}^1 x_2 + b_2^1)$, with b_1^1 and b_2^1 the bias of the neurons of the first hidden layer. The process can then be reproduces to obtain the output of the second hidden layer and the output layer.

The term "deep" in deep learning refers to the number of hidden layers in the neural network. Traditional neural networks, often referred to as shallow networks, typically have one or two hidden layers. In contrast, DNNs can have dozens or even hundreds of hidden layers. This depth allows DNNs to learn hierarchical representations of data, where higher-level features are derived from lower-level features [55].

DNNs have several advantages over traditional machine learning algorithms and shallow neural networks:

- **Feature Learning:** DNNs automatically learn features from raw data, reducing the need for manual feature extraction [38].
- **Modeling Complex Patterns:** The multiple layers in DNNs enable them to model complex, non-linear relationships in the data [55].
- **Scalability:** DNNs can handle large-scale datasets and are scalable to high-dimensional data [27].

Despite their advantages, DNNs also come with several challenges:

- **Computationally Intensive:** Training DNNs requires significant computational resources and time, especially for large networks [55].
- **Overfitting:** DNNs are prone to overfitting, particularly when the training data is limited or not representative of the real-world scenarios [63].
- **Hyperparameter Tuning:** Selecting the appropriate architecture and hyperparameters for a DNN is a complex and time-consuming task that often requires expertise and experimentation [4].

In the following sections, the different components involved in training a DNN will be studied, including the loss functions, the activation functions of the neurons, the optimization methods for the neural network, and the regularization methods to avoid overfitting.

Loss functions

A loss function may also be referred to as a cost function or an objective function. It is a mathematical function that describes the difference between predicted and true values built into the model for a machine learning algorithm. It tells how good or bad the model is performing. The primary purpose of such a loss function is to guide the training process of a machine learning model. It can then be adjusted, based on the output of the model, to fit the training data more accurately by minimizing the loss function.

Table 2.2: Presentation of Different Loss functions inspired from [27]

Loss Functions	Definition	Use cases
Mean Squared Error (MSE)	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ It measures the average of the squares of the errors, i.e., the average squared difference between the estimated values ($\hat{y}_i$) and the actual values (y_i).	• Regression problems
Mean Absolute Error (MAE)	$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$ MAE quantifies the average absolute difference between the actual (y_i) and predicted values ($\hat{y}_i$).	• Regression problems
Cross-Entropy Loss (Log Loss)	$\text{Cross-Entropy Loss} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$ Cross-Entropy Loss quantifies the difference between two probability distributions - the true labels and the predicted probabilities.	• Binary classification problems • Multi-class classification (with softmax)
Hinge Loss	$\text{Hinge Loss} = \frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i \cdot \hat{y}_i)$ Hinge Loss quantifies the error for classification tasks, particularly for SVMs, by penalizing misclassified points and those within the margin.	• Support Vector Machines (SVMs)
Huber Loss	$\text{Huber Loss} = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{for } \lvert y_i - \hat{y}_i\rvert \le \delta \\ \delta \cdot \lvert y_i - \hat{y}_i\rvert - \frac{1}{2}\delta^2 & \text{for } \lvert y_i - \hat{y}_i\rvert > \delta \end{cases}$ Huber Loss combines MSE and MAE to provide a robust measure that is less sensitive to outliers than MSE and more sensitive than MAE for small errors.	• Regression problems with outliers

Activation functions

Activation functions play a critical role in neural networks by introducing non-linearity, which enables the network to learn and model complex patterns in the data. Without activation functions, a neural network would essentially behave as a linear regression model, regardless of the number of layers. This section explores the most commonly used activation functions, their use cases, advantages, and disadvantages, supported by various research studies. ctivation functions determine the output of a neuron given an input or set of inputs. They are crucial because they allow neural networks to capture intricate structures in data, making them capable of solving non-trivial problems. Various studies have highlighted the importance of choosing appropriate activation functions to improve model performance and convergence speed [22, 38, 69]. table 2.3 presents the most commonly used activation functions.

There is a need to explain a common problem with activation functions, vanishing gradient. During backpropagation, depending on the activation function, the gradients used to update the weights can become very small, especially for deep networks. As a result, it

can slows down the learning process and potentially causing the network to stop learning altogether.

Table 2.3: Presentation of Different activation functions

Activation Functions	Definition	Use cases
Sigmoid [46]	$\sigma(x) = \frac{1}{1+e^{-x}}$ It squashes the input values to a range between 0 and 1.	• In the output layer for binary classification problems
TanH [46]	$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ It squashes the input values to a range between -1 and 1.	• Preferred over the sigmoid function for hidden layers in practice.
Rectified Linear Unit (ReLU) [46]	$\text{ReLU}(x) = \max(0, x)$	• Most widely used activation function in hidden layers of deep neural networks
Leaky ReLU [46]	$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$ where α is a small constant (e.g., 0.01).	• an attempt to fix the "dying ReLU" problem. For ReLu if x is negative, the output is null, so the output is "dying".
Parametric ReLU (PReLU) [50]	$\text{PReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$ where α is a learnable parameter.	• An enhancement of ReLU to allow the slope of the negative part to be learned during training
Exponential Linear Unit (ELU) [8]	$\text{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^{x}-1) & \text{if } x < 0 \end{cases}$ where α is a learnable parameter.	• Designed to bring the mean activation closer to zero and improve learning speed

Table 2.4: Comparison of the advantages and disadvantages of activation functions

Activation Functions	Advantages	Disadvantages
Sigmoid [46]	• Smooth gradient, preventing jumps in output values. • Output values bound between 0 and 1, making it suitable for probability estimation.	• Vanishing Gradient Problem • Sensitive to Outliers: Large positive or negative input values will be squashed to values near 1 or 0, respectively. This squashing effect can lead to a loss of information and make the network less robust to outliers in the data.
TanH [46]	• Zero-centered outputs, which can help in faster convergence [46]. • Steeper gradients than the sigmoid function.	• Still suffers from the vanishing gradient problem for large positive or negative input values [39, 46]
Rectified Linear Unit (ReLU) [46]	• Computationally efficient as it involves simple thresholding. • Helps mitigate the vanishing gradient problem [46, 50].	• Can suffer from the "dying ReLU" problem where neurons can become inactive and stop learning if they output zero for any input [46].
Leaky ReLU [46]	• Allows a small, non-zero gradient when the unit is not active.	• The choice of α is arbitrary and needs to be tuned.
Parametric ReLU (PReLU) [50]	• Adaptively learns the parameter α, potentially improving model performance. • Helps mitigate the "dying ReLU" problem more effectively than Leaky ReLU.	• Increases the number of parameters to be learned. • Computationally more complex than ReLU and Leaky ReLU [22].
Exponential Linear Unit (ELU) [8]	• Helps bring mean activation closer to zero, which speeds up learning. • Mitigates the vanishing gradient problem for negative inputs.	• More computationally expensive than ReLU. • The choice of α can impact performance and needs tuning [8].

Optimization Methods

Optimization in neural networks refers to the process of minimizing (or maximizing) the objective function, also known as the loss or cost function. This involves finding the best set of weights and biases that reduce the error between the predicted outputs and the actual outputs. The objective function $L(\theta)$, where θ represents the parameters of the network, is minimized using various optimization algorithms. These algorithms adjust the network parameters iteratively to find the optimal values that yield the best performance on the given task [27, 54]. The most widely used optimization methods in neural networks include:

Table 2.5: Optimization Methods in Machine Learning Part 1

Optimization Method	Definition
Gradient Descent [54]	**Equation:** $\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$ **Explanation:** • θ: Parameters to be optimized • η: Learning rate • $\nabla_\theta L(\theta_t)$: Gradient of the loss function with respect to θ at time step t **Definition:** Gradient Descent is an optimization algorithm that updates parameters in the opposite direction of the gradient of the loss function to minimize it.
Stochastic Gradient Descent (SGD) [5]	**Equation:** $\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t; x_i, y_i)$ **Explanation:** • x_i, y_i: A single training example • θ: Parameters to be optimized • η: Learning rate • $\nabla_\theta L(\theta_t; x_i, y_i)$: Gradient of the loss function with respect to θ using a single training example **Definition:** Stochastic Gradient Descent updates parameters using only one training example at each iteration, which can lead to faster convergence.
Momentum [48]	**Equation:** $v_{t+1} = \gamma v_t + \eta \nabla_\theta L(\theta_t)$ $\theta_{t+1} = \theta_t - v_{t+1}$ **Explanation:** • v_t: Velocity vector • γ: Momentum term (usually between 0 and 1) • η: Learning rate • $\nabla_\theta L(\theta_t)$: Gradient of the loss function with respect to θ at time step t **Definition:** Momentum accelerates gradient descent by considering the past gradients to smooth out the updates.
Adaptive Gradient Algorithm (AdaGrad) [73]	**Equation:** $\mathbf{s}_t = \mathbf{s}_{t-1} + g_t^2$ $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} g_t$ **Explanation:** • $\mathbf{s}_t$: Accumulated sum of squared gradients • g_t: Gradient of the loss function at time step t • η: Learning rate • ϵ: Small constant to prevent division by zero **Definition:** AdaGrad adapts the learning rate for each parameter individually, allowing larger updates for infrequent parameters and smaller updates for frequent parameters.

Table 2.6: Optimization Methods in Machine Learning Part 2

Optimization Method	Definition
Root Mean Square Propagation (RMSProp) [54]	**Equation:** $\mathbf{E}[g^2]_t = \gamma \mathbf{E}[g^2]_{t-1} + (1-\gamma) g_t^2$ $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbf{E}[g^2]_t + \epsilon}} g_t$ **Explanation:** • $\mathbf{E}[g^2]_t$: Exponential moving average of squared gradients • γ: Decay rate • g_t: Gradient of the loss function at time step t • η: Learning rate • ϵ: Small constant to prevent division by zero **Definition:** RMSProp adjusts the learning rate for each weight using an exponentially decaying average of past squared gradients.
Adam [35]	**Equation:** $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ $\theta_{t+1} = \theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ **Explanation:** • m_t: First moment (mean) of the gradients • v_t: Second moment (uncentered variance) of the gradients • $\hat{m}_t$: Bias-corrected first moment • $\hat{v}_t$: Bias-corrected second moment • β_1, β_2: Exponential decay rates for the moment estimates • g_t: Gradient of the loss function at time step t • η: Learning rate • ϵ: Small constant to prevent division by zero **Definition:** Adam combines the advantages of AdaGrad and RMSProp, using adaptive learning rates and incorporating estimates of first and second moments of the gradients.

Table 2.7: Comparison of Optimization Methods advantages and disadvantages

Optimization Method	Advantages	Disadvantages
Gradient Descent [54]	• Simple to implement. • Effective for convex problems.	• Can be slow for large datasets. • Prone to getting stuck in local minima.
Stochastic Gradient Descent (SGD) [5]	• Faster and more efficient for large datasets. • Helps in escaping local minima due to noise.	• May lead to unstable convergence. • Requires careful tuning of learning rate.
Momentum [48]	• Accelerates convergence. • Reduces oscillations in gradient updates.	• Can overshoot the minimum if not properly tuned. • Requires an additional hyperparameter.
Adaptive Gradient Algorithm (AdaGrad) [73]	• Automatically adapts the learning rate. • Performs well for sparse data.	• Learning rate can become too small over time. • Not suitable for non-convex problems.
Root Mean Square Propagation (RMSProp) [54]	• Prevents learning rate from decreasing too much. • Suitable for non-stationary problems.	• Requires tuning of decay hyperparameter. • Can be sensitive to learning rate.
Adam [35]	• Combines benefits of Momentum and RMSprop. • Well-suited for large datasets and non-convex problems.	• Can be computationally expensive. • Requires careful tuning of multiple hyperparameters.

Weight Initialization

Weight initialization in neural networks is a crucial step that can significantly impact the training process and the final performance of the model [39]. Proper initialization helps in speeding up the convergence of the training process and in avoiding issues like vanishing or exploding gradients. The possibilities for the training was *He normal initialization* and *Glorot uniform initialization*.

He normal initialization proposed by He et al. [22], is specifically designed for layers with ReLU (Rectified Linear Unit) activations. The weights are initialized using a normal distribution with a mean of 0 and a standard deviation of $\sqrt{\frac{2}{n}}$, where n is the number of input units in the layer. This method helps maintain the variance of the activations throughout the network, facilitating efficient training.

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right) \tag{2.11}$$

Advantages:

- Suitable for deep networks with ReLU activations.
- Helps maintain the variance of activations, avoiding vanishing gradients.

Glorot uniform initialization, also known as Xavier uniform initialization, was proposed by Glorot and Bengio [69]. It is designed to work well with sigmoid and tanh activation functions. The weights are initialized using a uniform distribution within the range $[-a, a]$, where $a = \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}$. Here, n_{in} and n_{out} are the number of input and output units, respectively. This method aims to keep the scale of the gradients roughly the same in all layers.

$$a = \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \quad W \sim \mathcal{U}(-a, a) \tag{2.12}$$

Advantages:

- Suitable for networks with sigmoid or tanh activations.
- Helps in maintaining the scale of gradients, which is crucial for effective training.

Regularization

Regularization in neural networks is a technique used to prevent overfitting, which occurs when a model learns the training data too well, including its noise and outliers, thus performing poorly on new, unseen data [27]. Regularization techniques add constraints or modifications to the learning process to improve the generalization capability of the model.

Dropout is a regularization technique where, during each training iteration, a subset of neurons is randomly set to zero (dropped out) [18]. This prevents the network from becoming too reliant on specific neurons, promoting redundancy and collaboration among neurons. During inference, dropout is not applied, and the full network is used, typically with scaled weights to account for the dropped neurons during training.

Regularization functions, such as L1 and L2 regularization, add a penalty term to the loss function based on the magnitude of the model parameters [75].

L1 regularization, also known as Lasso (Least Absolute Shrinkage and Selection Operator), adds the sum of the absolute values of the coefficients to the loss function. The equation for L1 regularization is:

$$\text{L1 Regularization:} \quad \lambda \sum_{i=1}^{n} |\theta_i| \tag{2.13}$$

where:

- λ is the regularization parameter that controls the strength of the regularization.
- θ_i are the model weights.
- n is the number of weights.

L2 regularization, also known as Ridge Regression, adds the sum of the squared values of the coefficients to the loss function. The equation for L2 regularization is:

$$\text{L2 Regularization:} \quad \lambda \sum_{i=1}^{n} \theta_i^2$$

where:

- λ is the regularization parameter that controls the strength of the regularization.
- θ_i are the model's weights.
- n is the number of weights.

L1 regularization adds the absolute values of the weights, encouraging sparsity, while L2 regularization adds the squared values of the weights, discouraging large weights.

Chapter 3

Related Work

In recent years, high-level control systems have been developed based on machine learning for lower-limb exoskeletons. In this background research, the various methodologies and contributions from the literature are reviewed, emphasizing machine learning techniques applied to this domain. This chapter present the specific Machine Learning techniques carried out in this study: Gaussian Process Regression, Neural Networks, Extreme Learning Machines, multiple regression methods, Nonlinear Autoregressive models, Neuro-fuzzy systems, and Support Vector Regression, among others. Specifically, it explains in what way each one advances the development of adaptive and personalized exoskeleton control systems.

3.1 Gaussian Process Regression Models

GPR is a very powerful machine learning technique for modeling and predicting continuous data. One of the central ideas of GPR is how, through a non-parametric approach, models of relationships between input variables and output variables can be built flexibly and accurately, even with nonlinear data.

Huang et al. [26] applied GPR combined with the Fourier series to model continuous kinematics at joints. The authors' method was based on utilizing the Fourier series to capture periodicity of gait cycles and leveraging the flexibility of GPR in modeling nonlinear relations between gait parameters and joint angles. The mathematical formulation of their approach can be summarized as follows:

1. Fourier Series Representation: They used a Fourier series to represent the periodic components of the gait cycle. The joint angle $q(\phi, \chi)$ as a function of time t can be expressed as:

$$q(\phi, \chi) = a_1(\chi) + \sum_{i=1}^{E} (a_{2i}(\chi)\cos(i\phi) + a_{2i+1}(\chi)\sin(i\phi)) = \mathbf{b}^T(\phi)\mathbf{A}(\chi)$$

   - $q(\phi, \chi)$: This represents the predicted joint kinematics as a function of phase (ϕ) and task variable (χ).
   - ϕ: The phase variable, typically representing the percent gait cycle.
   - χ: The task variable, which includes parameters such as speed and slope of walking.
   - $a_1(\chi)$, $a_{2i}(\chi)$, $a_{2i+1}(\chi)$: Fourier coefficients as functions of the task variable (χ).
   - E: The order of the Fourier series, selected to match the significant frequency content of human kinematics.

- $\cos(i\phi)$, $\sin(i\phi)$: The cosine and sine functions of the phase variable, which form the Fourier basis functions.
- $\mathbf{b}(\phi)$: The Fourier basis vector, defined as $[1, \cos(1\phi), \sin(1\phi), \ldots, \cos(E\phi), \sin(E\phi)]^T$.
- $\mathbf{A}(\chi)$: The Fourier coefficients vector, defined as $[a_1(\chi), a_2(\chi), \ldots, a_{2E+1}(\chi)]^T$.

2. Gaussian Process Regression: GPR was then used to model the relationship between the Fourier coefficients and the input gait parameters whit are walking speed and slope. The GP defined to expressed the Fourier coefficients are:

$$\hat{f}_k(\chi) \sim GP\left(\mu_{\hat{f}_k}(\chi), k_{\hat{f}_k}(\chi, \chi')\right)$$

where:

- $\hat{f}_k(\chi)$: The predicted Fourier coefficient for the k-th dimension as a function of the task variable χ.
- $\mu_{\hat{f}_k}(\chi)$: The mean function of the Gaussian Process for the k-th Fourier coefficient.
- $k_{\hat{f}_k}(\chi, \chi')$: The kernel (covariance) function that defines the covariance between two task variables χ and χ'.
- D_k: The training dataset for the k-th Fourier coefficient.
- χ_*: The new task input for prediction.
- $k_{\hat{f}_k} = k_{\hat{f}_k}(D_k, \chi_*)$: The kernel vector.
- $K_{\hat{f}_k}$: The kernel matrix with entries $K_{ij}^{\hat{f}_k} = k_{\hat{f}_k}(\chi_i, \chi_j)$.
- σ_f: The signal standard deviation.
- σ_l: The characteristic length scale.

3. Objective: An essential element in their approach was the development of a model capable of accurately predicting trajectories of joints at various walking speeds and slopes. Their approach combined periodicity representation in the gait cycle with the flexibility of GPR in a nonparametric sense, thus providing a robust solution for modeling human gait in an adaptive way.

For a new task input χ_*, the prediction for the k-th Fourier coefficient is given by:

$$p(\hat{f}_k(\chi_*)|D_k, \chi_*) = \mathcal{N}\left(\mu_{\hat{f}_k}(\chi_*), \sigma^2_{\hat{f}_k}(\chi_*)\right) \tag{3.1}$$

where:

- D_k: The training dataset for the k-th Fourier coefficient.
- $\mu_{\hat{f}_k}(\chi_*)$: The mean prediction at the new task input χ_*.
- $\sigma^2_{\hat{f}_k}(\chi_*)$: The variance of the prediction at the new task input χ_*.

The mean and variance predictions are computed using a kernel vector $k_{\hat{f}_k} = k_{\hat{f}_k}(D_k, \chi_*)$ and a kernel matrix $K_{\hat{f}_k}$ with entries $K_{ij}^{\hat{f}_k} = k_{\hat{f}_k}(\chi_i, \chi_j)$:

$$\mu_{\hat{f}_k}(\chi_*) = k_{\hat{f}_k}^T K_{\hat{f}_k}^{-1} D_k \tag{3.2}$$

$$\sigma^2_{\hat{f}_k}(\chi_*) = k_{\hat{f}_k}(\chi_*, \chi_*) - k^T_{\hat{f}_k} K^{-1}_{\hat{f}_k} k_{\hat{f}_k} \tag{3.3}$$

The kernel function used is the exponential kernel with automatic relevance determination (ARD):

$$k_{\hat{f}_k}(\chi, \chi') = \sigma_f^2 \exp\left(-\frac{1}{2}(\chi - \chi')^T(\chi - \chi')/\sigma_l^2\right) \tag{3.4}$$

where:

- σ_f: The signal standard deviation.
- σ_l: The characteristic length scale.

The hyperparameters $\theta = [\sigma_f, \sigma_l]$ are determined by maximizing the log marginal likelihood function:

$$\log P(D_k|x_{ob}, \Theta) = -\frac{1}{2}\left(D_k^T K^{-1}_{\hat{f}_k} D_k + \log|K_{\hat{f}_k}| + n\log 2\pi\right) \tag{3.5}$$

where n is the number of recorded tasks.

The results show major decreases, in terms of root mean squared error and maximum error, respectively, after individualization of the model. Precisely, the average reduction in RMSE/Max-Error at the hip, knee, and ankle joints comes to 2.94°/5.059°, 1.822°/5.82°, and 1.156°/4.51° degrees, respectively. The Gaussian Process Enhanced Fourier Series (GPEFS) model is very promising for the powered prosthesis continuous control paradigm, especially for creating a unified controller across the whole gait cycle. This approach avoids the discretization of the control paradigms, which requires manual tuning of the parameters for every single task.

While the transformation into Fourier coefficient space reduces computational load, this method still requires some huge computational resources, especially in real-time applications. Some methods related to this work show relatively good predictive performance but require extensive computational resources, which are really not suitable for real-time applications. On the other hand, the GPEFS seeks to balance predictive accuracy and computational efficiency; however, it is challenging to reach such a balance.

Eslamy and Rastgaar [14] estimated multi-joint leg moments while walking using GPR. In that direction, their aim was to give a model able to predict, with high accuracy, the required joint moments at different phases of the gait to improve the control of exoskeletons. Their approach uses the following inputs and outputs:

- Inputs: The primary inputs to the GPR model were thigh and shank angles. These angles are one of the important parameters, which carry information about the kinematic state of the lower limb related to walking.
- Outputs: The output variables from the GPR model were the estimated joint moments. These are forces which are required at joints such as the hip, knee, and ankle to create the desired movements during various phases of the gait cycle.

They essentially worked towards predicting the joint moments from the kinematic data input. The potential of this capability in exoskeleton control is huge because very fine-tuned adjustments in applied assistive forces by the device would be achieved and, subsequently, so the user's gait stability and comfort. table 3.1 presents the results of the model:

Table 3.1: Results of Multi-Joint Leg Moment Estimation from [14]

Walking Speed	RMS Errors [Nm/kg]	MAEs [Nm/kg]
Hip Moment Estimation Using Thigh Angles		
0.5 m/s	0.13±0.05	0.10±0.04
1.0 m/s	0.13±0.05	0.10±0.04
1.5 m/s	0.15±0.05	0.11±0.04
Knee Moment Estimation Using Thigh Angles		
0.5 m/s	0.13±0.05	0.10±0.04
1.0 m/s	0.13±0.06	0.10±0.05
1.5 m/s	0.13±0.06	0.09±0.05
Ankle Moment Estimation Using Thigh Angles		
0.5 m/s	0.13±0.05	0.10±0.04
1.0 m/s	0.12±0.04	0.08±0.02
1.5 m/s	0.10±0.04	0.07±0.03
Ankle Moment Estimation Using Shank Angles		
0.5 m/s	0.15±0.05	0.11±0.03
1.0 m/s	0.10±0.02	0.07±0.02
1.5 m/s	0.95±0.03	0.97±0.01

Some disadvantages from this method are observed by the authors. The estimation quality tends to be lower at lower speeds, especially for the knee joint. This indicates a potential limitation in the robustness of the method across different walking speeds. The method's accuracy is highly dependent on the input signals. Variability in the input quality can significantly affect the estimation results.

3.2 Neural Networks Models

Neural networks, especially the BNN (Back Propagation Neural Networks) and Long Short Term Memory (LSTM), have a good promise in predicting joint movements for the control of exoskeletons.

3.2.1 Back Propagation Neural Networks

A BNN is an artificial neural network in which the error, or the difference between the predicted and actual output, is passed backward in order to update the weights [68]. It is through this process, backpropagation, that the network will learn how to choose with the best weights that ensure a minimized error.

Sá Cunha et al. [47] used BNN to estimate subject-specific knee and hip joint angles . The authors trained the BNN on data containing a number of subject-specific parameters, including age, height, weight, and gait speed as represented in fig. 3.1. It is easy to see how through its deep architecture the neural network could learn mappings between such inputs and the desired joint angles so as to output personalized gait patterns adapting to characteristics of individual users.

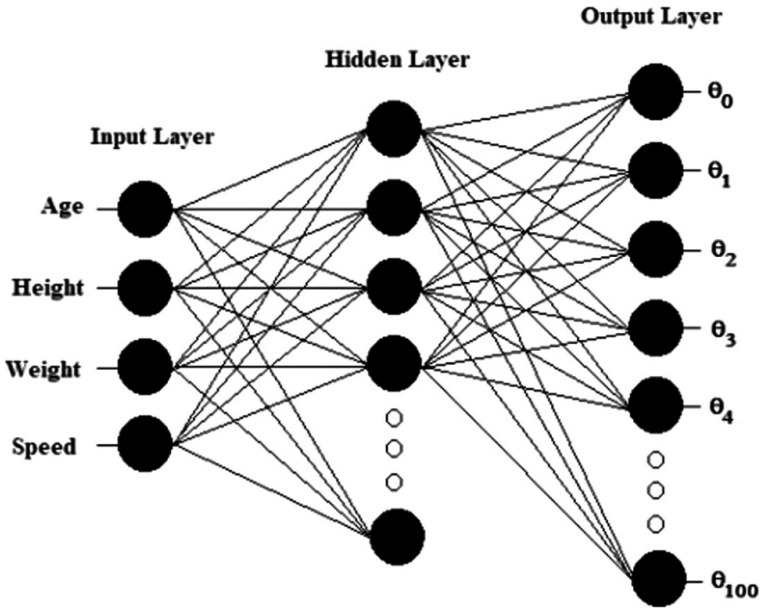


Figure 3.1: BNN representation from [47]. As inputs there is the age, height, Weight and walking speed of the subject as inputs, and as outputs there are the different points of the joint trajectory taken at 0, 1, 2, 3, ... 100 percent of the gait cycle.

The mean on the MSE of the results is 0.002. The methods used in this study are applicable in the fields of biomechanics and medicine, particularly for the detection of gait pathologies and rehabilitation. By generating subject-specific joint angle reference profiles based on parameters such as height, weight, age, and walking speed, the study provides more accurate comparisons and diagnoses compared to using generalized healthy reference curves. The main disadvantage of the BNN method noted in the study is the lack of smoothness in the generated curves for subjects at the edge of the training dataset. While BNNs showed higher accuracy in most cases, they struggled to produce smooth joint angle curves for these subjects.

3.2.2 Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data [20]. Unlike traditional RNNs, LSTMs use a unique structure with gates that control the flow of information, allowing them to remember and forget information over long sequences.

Liang et al. [41] used LSTM networks for online adaptive trajectory generation in lower limb exoskeletons. The main motivation behind using LSTMs in their case was that they can effectively model temporal dependencies between cycles of gait, as the variables to predict depend on the time and LSTM are efficient in those case, which is a core component of being able to predict future positions of joints from past movement. The inputs to this LSTM network contained past joint angles and velocities. This sequential data fed back information about the current and previous states of the joints to the network. The outputs of the LSTM network were the predicted joint angles at the moment there are needed. Such were then used to create adaptive trajectories for the exoskeleton, enabling smooth and natural movements. These advantages that LSTMs provide for this include handling the sequential nature of gait data, capturing long-term dependencies robustly, and real-time prediction ability. The exploitation of these advantages would make the LSTM-based system quickly adapt

to changes in the user's gait, hence improving control and performance of the exoskeleton.

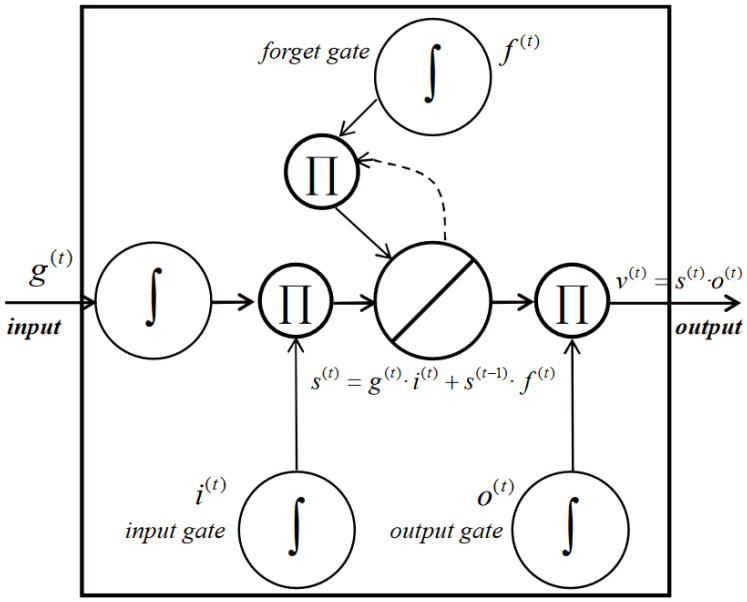


Figure 3.2: Structure of an LSTM Cell from [41]. This figure illustrates the architecture of a Long Short-Term Memory (LSTM) cell, which includes the following components: **Input Gate ($i^{(t)}$)**: Controls the extent to which a new value flows into the cell. **Forget Gate ($f^{(t)}$)**: Determines what portion of the cell state should be forgotten. **Cell State ($s^{(t)}$)**: Represents the memory of the cell, combining the previous cell state $s^{(t-1)}$, the current input $g^{(t)}$, and the forget gate $f^{(t)}$. **Output Gate ($o^{(t)}$)**: Modulates the cell state to produce the output $v^{(t)}$. **Activation Functions (σ and** tanh**)**: Non-linear functions applied to the gates and cell state.
The formula inside the cell indicates how the cell state is updated: $s^{(t)} = g^{(t)} \cdot i^{(t)} + s^{(t-1)} \cdot f^{(t)}$ And the final output of the cell is calculated as: $v^{(t)} = s^{(t)} \cdot o^{(t)}$.

The mean RMSE obtain at the end and there are 1.21° for the Hip and 2.23° for the knee. Despite the promising results, the method has several disadvantages. The LSTM-based approach requires significant computational resources for training and real-time adaptation, which may limit its applicability in resource-constrained environments. he effectiveness of the method is highly dependent on the quality and quantity of the training data, which may not be readily available in all clinical settings.

3.3 Extreme Learning Machine (ELM)

ELMs are a much faster in training and in terms of computation time in prediction and efficient alternative to traditional neural networks [25]. The extreme learning machine consists of a single hidden layer feedforward network where the weights get assigned randomly. It reduces drastically the training time compared to conventional neural networks. An important notion here would be the setting of random weights between the input and the hidden layers. Those weights are fixed, and only the weights between the hidden and output layers are learned.

An autoencoder is an artificial neural network architectural variant used to learn efficient codings, here a representation of data from one dimnesional space to another (smaller), in an unsupervised manner [24]. It consists of an encoder which maps the input data into a lower dimensional space and a decoder that tries to reconstruct the input data from the encoded

representation. Autoencoders are applied in unsupervised learning for the dimensionality reduction and feature learning.

He et al. [23] combined ELMs with autoencoders to enhance the modeling of subject-specific gait profiles. The primary objective of their study was to develop a fast and accurate method for generating rehabilitation gait trajectories for lower extremity exoskeletons. The goal was to create a system that could quickly adapt to the user's gait patterns and provide personalized assistance. ELM was chosen due to its fast learning speed and good generalization performance, with the capability to handle a large amount of data in an effective way, so it is suitable for real-time applications like exoskeleton control. By combining ELM with autoencoders, the authors enhanced the model's ability to extract relevant features from the input data, improving the accuracy of the predictions.

The inputs to the ELM were various features extracted from the user's gait data, such as joint angles, velocities, and accelerations. The outputs of the ELM were the predicted angle joint trajectories, which were used to control the exoskeleton. The model representation can be seen in fig. 3.3.

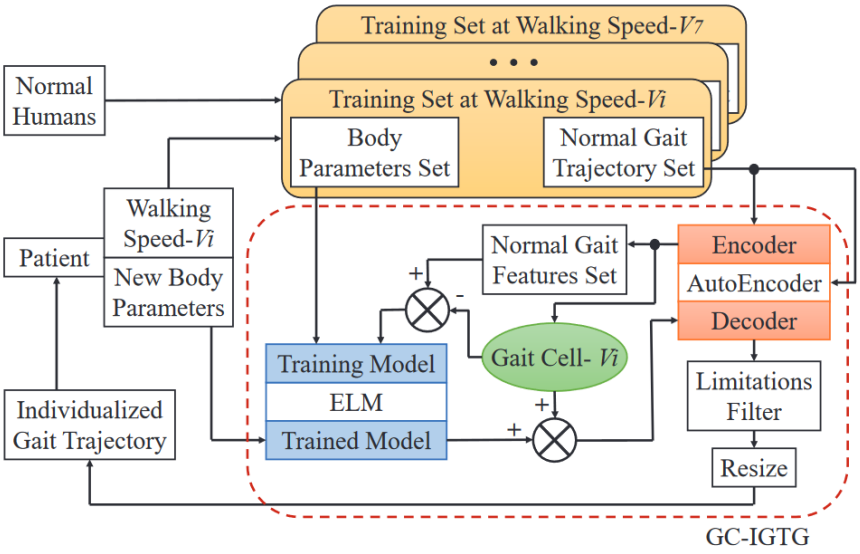


Figure 3.3: The framework of the Gait Cell based Individualized Gait Trajectory Generation (GC-IGTG) algorithm. The GC-IGTG method employs extreme learning machines (ELM) and AutoEncoders to generate individualized gait trajectories for lower limb exoskeletons used in stroke rehabilitation. The process involves collecting body parameters and normal gait trajectory data from normal humans at various walking speeds to create a training set. This training set is then used to train the ELM model, which, in conjunction with the AutoEncoder, generates individualized gait trajectories based on the new body parameters and walking speed of the patient. The algorithm also includes a limitations filter to ensure the generated trajectories are safe and feasible for practical use.

The deviation error (Cocl), which is the absolute value of the error on one gait length between predicted and real trajectory divided by the real trajectory, for the one-cycle gait length was less than 4%, indicating high precision. he GC-IGTG algorithm can be used to enhance the control and effectiveness of lower limb exoskeletons, providing customized rehabilitation exercises for stroke patients. The algorithm requires significant computational resources for training and real-time adaptation, which may limit its applicability in resource-constrained environments.

3.4 Multiples Regressions

One of the most used statistical techniques in modeling the relationship between a dependent variable and multiple independent variables is multiple regression. In gait analysis, multiple regression techniques are applied in the prediction of gait parameters, such as joint angles and velocities, using various demographic and biomechanical inputs [44].

Moissenet et al. [44] recorded the data for walking speed, gender, age, and BMI of each subject and their corresponding joint angles during gait cycles. The multiple regression model was defined to predict key points of the joint angle trajectories using the following equation:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon \tag{3.6}$$

where Y represents the key points of the joint angles, $X_1, X_2, \ldots, X_n$ are the independent variables (walking speed, gender, age, BMI), $\beta_0, \beta_1, \ldots, \beta_n$ The regression factors.

Specific key points within the gait cycle, corresponding to critical events such as heel strike, mid-stance, and toe-off, were identified for each joint angle. It is the multiple regression analysis that predicted these key point values with demographic and biomechanical inputs. Spline interpolation was used to generate continuous joint angle trajectories from the predicted key points, ensuring smooth transitions between them.

The outputs are continuous trajectories of joint angles, generated through the interpolation of key points predicted by the multiple regression model. These outputs are used to personalize the control strategies for lower limb exoskeletons, enhancing their adaptability and effectiveness.

Root mean square error (RMSE) for timing and angle at the hip, knee, and ankle joints were as follows:

- Hip: 1.42% of gait cycle, 5.55° for angles.
- Knee: 1.65% of gait cycle, 4.79° for angles.
- Ankle: 1.65% of gait cycle, 3.36° for angles.

The regression model showed increased errors at walking speeds below 0.2 and above 0.7 dimensionless walking speed, indicating that the model may not be suitable for very slow or very fast walking speeds. The study suggested that a nonlinear approach might be required for accurately describing transition phases at these walking speeds.

3.5 Nonlinear Autoregressive Models

Nonlinear Autoregressive models are time-series models used for the forecasting of future values with respect to their past values, capturing the nonlinear relationship that exists within them. Therefore, they can be applied in modeling dynamic systems such as human gait [16].

Farmer et al. [16] sought to design a model that could predict continuous ankle kinematics of powered transtibial prostheses users, allowing real-time control and improving the walking experience. Here are the inputs and outputs of their model represented in fig. 2.3:

- Inputs: Past values of the time series 10 - 100 ms before t(e.g., joint angles, velocities, EMG signals).
- Outputs: Predicted future values of the time series at time t (e.g., joint angles or velocities).

The Nonlinear autoregressive model can be define as:

$$y(t) = f(y(t-1), y(t-2), \ldots, y(t-n)) + \epsilon(t)$$

where $y(t)$ is the time series value at time t, f is a nonlinear function, and $\epsilon(t)$ is the error term. They must then train the model using historical data to fit the nonlinear function f using techniques such as neural networks or polynomial regression.

The nonlinear autoregressive model successfully predicted the ankle kinematics of the prosthesis during level treadmill walking in three transtibial amputees. For prediction intervals up to 150 ms, the root mean square error (RMSE) between the predicted and actual ankle angle ranged from 0.7° to 6.3°, depending on the prediction interval and the subject.

Some weaknesses fo the moden can be observed. Model performance varied across subjects, indicating that the approach may need individual calibration to achieve optimal results. While the model performed well for prediction intervals up to 150 ms, errors increased systematically with larger prediction intervals. The model's applicability to a larger cohort of amputee subjects and different types of prosthetic devices remains to be validated.

3.6 Neuro-fuzzy systems

Neurofuzzy systems combine the human-like reasoning style of fuzzy systems with the learning and connectionist structure from neural networks [29]. Such a system will hence be able to inherit the intrinsic advantages of both: fuzzy systems to deal with uncertainty and imprecision, and neural networks to learn from data. Here are the key features of neuro-fuzzy systems [29]:

- Fuzzy Inference System (FIS): This corresponds to the fuzzy rules, membership functions, and the reasoning mechanism itself. Membership functions as in fig. 2.2 define how each point in the input space is mapped to a degree of membership between 0 and 1 [29]. They quantify the fuzziness and represent the degree to which a given input belongs to a fuzzy set.
- Neural Network Structure: Parameters of the FIS are adjusted using neural network learning algorithms.
- Hybrid Learning: Combines gradient descent for parameter optimization and least squares estimation for rule parameter identification.

Kiguchi et al. [34] developed a neuro-fuzzy control of a robotic exoskeleton using EMG signals. The methodology can be divided into different steps. The experimental setup included the collection of EMG signals from the user's muscles, providing real-time information about intended movements by the user. At the core of their approach was a fuzzy inference system that translated EMG signals into control signals for the exoskeleton, here the angles of the joints. In essence, the FIS comprised fuzzy rules and membership functions that map the EMG inputs onto the control output. Optimization of membership function parameters and the fuzzy inference system's rule weights was done using neural network learning techniques. In other words, these parameters are tuned in order to improve the accuracy of the translation by the system from EMG signals into control commands.

The method has his benefits [34]. This would make the control exoskeleton more robust and flexible due to learning based on the neuro-fuzzy system; adaptation to a different user or different and variable conditions would be possible. The system could process EMG signals in real-time, thus provide instant responses to the movements by the user. Introducing fuzzy logic into this system would handle the uncertainties and variabilities existing in EMG signals well.

3.7 Support Vector Regression

Support Vector Regression is another algorithm for machine learning that addresses the problem of regression. It extends the concepts of Support Vector Machines to handle continuous valued output variables, hence making it quite suitable for real-valued outputs [62]. The key concepts of this algorithm are:

- Margin of Tolerance: SVR finds a function that deviates from the actual observed values by no more than ϵ (epsilon-insensitive loss function).
- Support Vectors: Only the data points lying outside the tolerance margin, the so-called support vectors, contribute to a model, hence making it robust against outliers.
- Kernel Trick: Nonlinear relations in SVR are taken care of by kernel functions, which transform the input space to higher dimensions for linear regression.

Dey et al. [9] further applied this to the continuous prediction of ankle angles and moments during walking using SVR. The inputs are the values of hip and knee angles and velocities the instant before. The output are the next predicted ankle angles and moments for controlling active ankle prostheses.

SVR makes robust predictions even with noisy and complex data. The kernel functions will capture the nonlinear relationships, and it provides good generalization to new data. table 3.2 present the results of the different combinations of inputs and output and their results. The R^2 results are very good, demonstrating that the model used has captured a good part of the variability in the data. As a reminder, the maximum value of R^2 is 1, which is the best value. The rest of the RMSE results are interpreted in the article as satisfactory.

The experimentation and model possess certain limitations and disadvantages. The algorithm was validated on level-walking datasets from only one healthy subject. There possibility to expand it to different subjects need to be experiment and validate. Calculating velocities using the finite difference method in real-time can lead to lags.

Table 3.2: Prediction Accuracy of the SVR Model for Different Input Cases from [9].

Case	Inputs	Output	Mean R^2	RMSE
I	$\theta_{\text{hip}}, \theta_{\text{knee}}$	θ_{ankle}	0.95	2.17°
		τ_{ankle}	0.95	0.11 Nm/kg
II	$\theta_{\text{hip}}, \theta_{\text{knee}}, \dot{\theta}_{\text{knee}}$	θ_{ankle}	0.97	1.67°
		τ_{ankle}	0.96	0.10 Nm/kg
III	$\theta_{\text{hip}}, \theta_{\text{knee}}, \dot{\theta}_{\text{knee}}, \dot{\theta}_{\text{hip}}$	θ_{ankle}	0.98	1.29°
		τ_{ankle}	0.97	0.08 Nm/kg

3.8 Comparison of Methods

Each of these methods has its strengths and relates to different aspects of high-level control for lower limb exoskeletons. GPR and LSTM networks show very good performance with sequential and time-series data and thus are appropriate for real-time gait prediction. Neural networks, like BNN and ELM, bring flexibility and efficiency while learning complex mappings from a wide range of input parameters. It is in these regression techniques and autoregressive models that robust frameworks can be found for the inclusion of demographic and biomechanical variables into the control system to enhance both personalization and

adaptability. At the other end are hybrid approaches, like neuro-fuzzy systems, which amalgamate the best of neural networks and fuzzy logic to handle effectively the uncertainties in the input data.

The most common shortcomings of these methods are adaptability to any subject and online use of the controller. Each of the methods presented has one or other of these difficulties. The controller's difference for this work will be its adaptability to any subject.

Chapter 4

Design and Implementation

This thesis, in collaboration with the TUM DASH student initiative, focuses on the design of a high-level controller. The objective of the controller is, for each active joint of the DASH exoskeleton, to produce its trajectory for a complete gait cycle, for walking on a flat surface. In this context, the controller must meet the following requirements:

1. The exoskeleton has six DoFs. Each joint has an individual joint controller tracking a desired position. There are three DoF per leg: hip abduction, hip flexion and knee flexion.
2. The controller must generate a trajectory for a complete gait cycle for each DoF, specifying both position and speed.
3. The controller must have as input the kinematic state (see section 2.3) of the exoskeleton.
4. The computation time the controller must be appropriate to anticipate potential integration into the DASH exoskeleton.

Taking these requirements into account, Machine Learning models were chosen to develop this controller.

In the subsequent sections, we will first introduce the controller designed for this study, outlining its components and their respective functions. Following this, we will provide an in-depth examination of the controller's implementation, which encompasses the development of the database, the key point search process, and the training methodologies employed for the two types of Machine Learning models utilized in this research: Neural Networks and Gaussian Process Regression.

4.1 Controller Overview

The controller, of which a diagram can be seen fig. 4.1, is divided into two main components:

- Gait phase estimation
- Trajectory generation

The repeatability of the standard human gait cycle allows for the state of various joints to be used at any time to estimate an individual's phase within the current gait cycle. Thus, the gait phase estimation component estimates the current gait phase as percentage of the current cycle of the system (both pilot and exoskeleton).

The trajectory generation operates on the principle that distinct key points exist in the trajectories of each joint. In the original concept, each DoF had an individual trajectory generator. Their role was to generate key points specific to the DoF to which they were attached. These key points are extremums and curvature inversion / inflection points, with the trajectory obtained by interpolating these points. To enhance the performance of this interpolation, additional points, in between the aforementioned key points, are selected. These points are referred to as key points. can be caused by a number of factors. Here, following on from studies such as that by Moissenet [44] or Chehab [7], the factors studied here will be walking speed and pilot parameters.

A neural network is employed to learn the relationships between the input parameters (pilot parameters (height, weight, age, gender), walking speed, and DoF position) and the corresponding gait phase and the outputs, namely the key points. Finally, interpolation is performed on these key points to obtain the desired trajectory. The generated trajectories are normalized, expressed in terms of gait percentage rather than time. However, since the rest of the system requires trajectories expressed as a function of time, it is essential to estimate the gait period to convert the trajectories from a gait percentage basis to a time-based expression before passing it to the rest of the system.

4.1.1 Gait Phase Estimation

To ensure continuity with the current position of the joints/DoF when generating a trajectory, the controller uses the DoF's current position and the corresponding gait percentage as inputs. This necessitated the development of an intent recognition module to determine the percentage of the current gait.

Building on the work of Simon Barnikol [1], which established that the gait phase could be estimated using the DoF's position and speed, this thesis extends the concept to the entire set of DoFs applicable to the TUM DASH exoskeleton. The hypothesis is that utilizing a broader combination of parameters corresponding to the same gait phase will improve the accuracy of gait phase estimation.

Inspired by Barnikol's work [1], the machine learning model employed is a Gaussian Process Regression, as depicted in fig. 4.2. This model facilitates the expression of a gait phase value along with its margin of error. The training of this type of Machine Learning model will be looked into further in section 4.2.4.

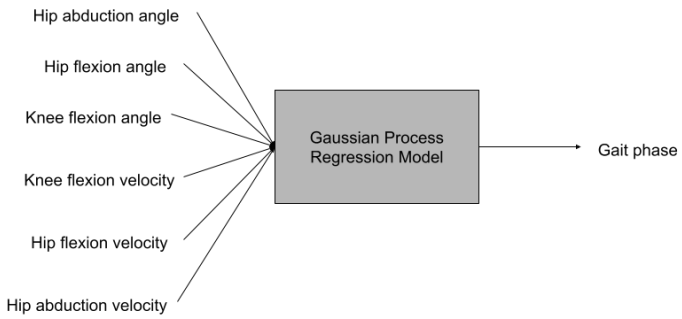


Figure 4.2: Concept of model for gait phase estimation. The principle is to evaluate, from the state of the pilot and the exoskeleton at time t, what percentage of the gait the pilot is in. In this case, the state at time t corresponds to the speed and position of each of the DoFs. Based on all this information, we consider it uniquely possible to determine at which gait phase we are.

4.1.2 Key points selection

As discussed in chapter 2, distinct shapes exist in the trajectories of the leg joints. These shapes are intricately linked to the characteristics of each DoF's motion. The initial conjecture from which the subsequent analysis stems is the identification of key points, namely the extrema of these trajectories. Those extrema corresponds to the position during which the speed of the joint DoF is zero. Studies as [7] established the existence of a fixed number of extrema, depending on the DoF. They also give an interval of gait percentages to find those extrema.

Following this premise, another crucial step involves establishing the existence of inflection points within the framework of this thesis. It is postulated that between any two consecutive extrema, an inflection point is present, thereby signifying a transition in the movement profile.

The determination of inflection points augments the list of key points, enriching the specificity of the trajectory analysis. They corresponds to the points where the speed of the joint DoF will be maxed and the acceleration will be null. Furthermore, to provide a comprehensive foundation for the interpolation methodology, the initiation and end points of each curve are specified as boundary conditions. Complementary to these constraints, supplementary points are strategically chosen to refine the interpolation accuracy. These additional points are selected based on their correlation with gait percentage and are expressed as a function of the extremum gait percentage. Notably, the selection of these points varies across the individual DoFs, necessitating a tailored approach for each joint.

The optimization of interpolation performance is guided by a systematic process involving trial and error. By iteratively adding points in regions where approximation errors are most pronounced, the interpolation accuracy is systematically enhanced. This rigorous methodology ensures a comprehensive evaluation of the trajectory characteristics and facilitates the refinement of the interpolation technique.

A maximum of 12 key points per degree of freedom (DoF) was established to constrain the neural network's output. The details of each key points for the different DoFs will be seen in the section 4.2.

4.1.3 Neural Network model

Neural networks form the foundation of the trajectory generation. Drawing inspiration from prior works like [44] and [37], which demonstrated the feasibility of establishing regressions between selected key points on the curve and factors such as walking speed and individual parameters (age, gender, weight, height), the concept of this thesis is to employ a neural network as a model for multiple regressions. The adoption of a deep neural network was chosen due to its capability to learn intricate and non-linear relationships effectively.

This approach aims to establish correlations between inputs—such as walking speed, individual pilot parameters (age, gender, weight, height), and inputs ensuring trajectory continuity (gait phase, joint position)—and outputs, which correspond to the key points on the trajectory. Each key point consists of a pair indicating gait phase and joint position, necessitating two outputs per key point. Therefore, for 12 key points, the neural network encompasses 24 outputs, illustrated in fig. 4.3.It is possible to take a different number of key points per DoF. But this would require each DoF to redetermine the hyperparameters of each model linked to it, since the structure of the model would vary according to the number of outputs. Here, the assumption is that by fixing the number of key points for all DoFs, it is possible to determine the model hyperparameters for one DoF and then reuse them for the others, since the structure remains similar.

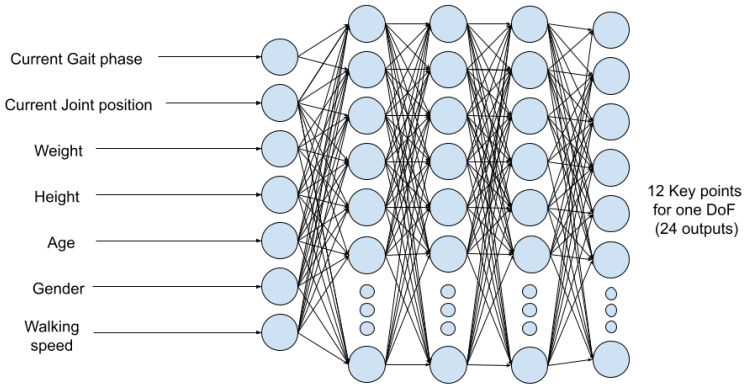


Figure 4.3: Concept for the trajectory generation for one DoF. The model is designed to accept inputs including walking speed and individual parameters such as age, height, and weight. It generates 24 output values, organized in pairs (gait phase, DoF position), which collectively define each of the 12 key points to be generated.

4.1.4 Gait Period Estimation

As previously outlined, the generated trajectories are normalized, specifically expressed as a function of gait percentage. However, for the trajectories to align temporally within the system, they must be articulated as a function of time. Therefore, there is a necessity to continuously determine the gait period, the duration of one complete gait cycle.

Building upon the findings of works [72] and [74], where gait period emerges as a key parameter of gait, a model illustrated in fig. 4.4 is developed. This model utilizes inputs such as individual parameters (age, gender, height, weight) and the walking speed to estimate the gait period. The selection of Gaussian Process Regression for this estimation is also inspired by these studies, as it provides a means to quantify the uncertainty associated with the obtained gait period. The general training method of the Gaussian Process Regression will be seen in section 4.2.4.

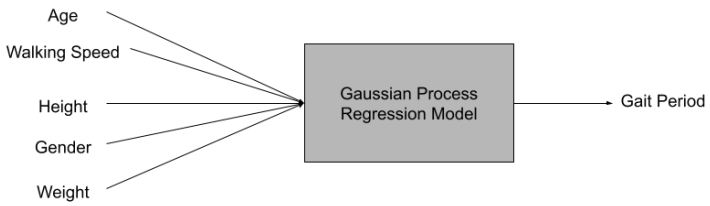


Figure 4.4: Concept for gait period estimation. Employing a Gaussian Process Regression model, chosen for its ability to capture complex, non-linear relationships, the study estimates the gait period using driver parameters such as age, gender, height, weight, and walking speed which is selected by the pilot.

4.2 Controller Implementation

In this section, we delve into the implementation details of the controller, emphasizing the processes involved in data collection for training the various Machine Learning (ML) models in section 4.2.1. We will also elaborate on the specifics of the key points, distinguishing them by Degrees of Freedom (DoF). Lastly, we will outline the training plan for the ML models.

4.2.1 Databases Selection

Since the components of this controller are based on machine learning models, their implementation necessitates appropriate training data. Specifically, the required data consists of walking gait trajectories on flat surfaces. Each trajectory must be associated with the subject from whom it was measured. For each subject, essential information, including age, gender, height, and weight, must be provided.

In the context of this thesis, collaborating with the students of the TUM DASH initiative to acquire a sufficient quantity of data for training various models poses significant challenges in terms of equipment and time. The collection of these trajectories requires motion capture technology. After recording the motion capture videos, the positions of the trackers must be interpreted over time, and the joint angles deduced using inverse kinematics. This process is both lengthy and labor-intensive.

To circumvent this extensive process, several online databases provide direct access to the required trajectories, specifically the joint angles for various DoFs. These databases generally include the essential subject information needed for the defined models, along with additional data such as torque per DoF and EMG measurements during gait.

The objective of the Controller is to generate trajectories for each joint to replicate the gait of a healthy person at a given speed on a flat surface. Therefore, the databases to be selected must meet the following conditions:

- The measurements must come from healthy persons
- The databases must provide the computed trajectories for the desired DoFs (hip abduction, hip flexion and knee flexion) and not just the motions captures datas, sensors datas.
- There must be experiments with multiples walking speeds on plane surfaces

The goal was to extract the desired trajectories from these databases. A walking experiment is defined by two conditions: the walking speed and the incline of the surface on which the subject is walking. For each experiment multiple gait cycle can be provided, or a mean on all gait cycles measured during the experiment as it is the case with [17]. The databases utilized in this thesis are:

- [31] of K. Embry et al, which have 10 subjects with height ranging from 1.57 to 1.86m, mass ranging from 47.8 to 75.0 kg and age ranging from 19 to 27 years old. There are 3 experiments per subject only those with a null incline are considered with walking speed 0.8, 1.0, 1.2 m/s. Each experiment has 45 gait cycles are provided.
- [17] of Fukuchi et al, with a population of subjects, including 24 young adults (age 27.6 ± 4.4 years, height 171.1 ± 10.5 cm, and mass 68.4 ± 12.2 kg) and 18 older adults (age 62.7 ± 8.0 years, height 161.8 ± 9.5 cm, and mass 66.9 ± 10.1 kg). There are 12 experiments per subject, with walking speed in the range of 0.23 to 2.23 m/s, depending on the comfort speed of the subject.
- [45] of Moreira et al, with a population of 16 volunteers with height ranging from 1.51 to 1.83 m, mass ranging from 52.0 to 83.7 kg and age ranging from 20 to 28 years old. There are 7 experiments per person, with a walking speed ranging from 0.28 to 1.1 m/s, with 10 trial (10 gait cycle per experiment).

Subject data, walking experiment data and required joint trajectories were extracted and stored in a data structure according to fig. 4.5. They were then subjected to additional filtering related to key points, which will be discussed in the subsequent section.

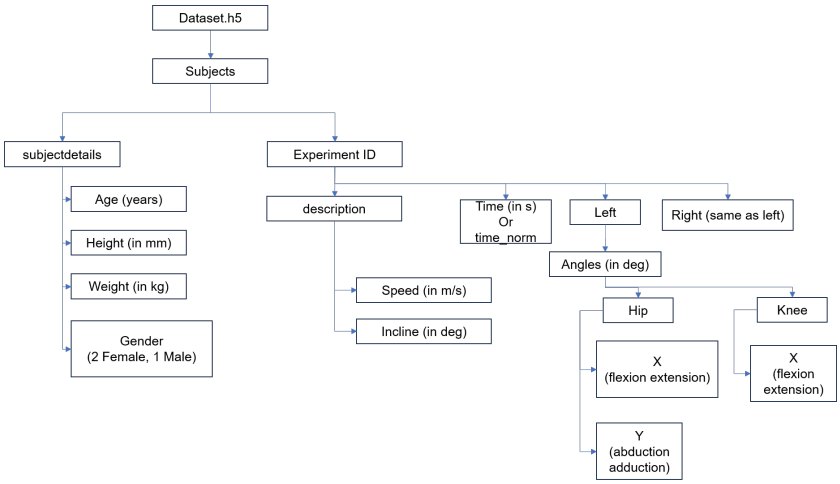


Figure 4.5: The data is organized in the dataset.h5 according to the subjects from whom the trajectories were measured. For each subject, relevant details such as age, gender, height, and weight are stored in a structure labeled subjectdetails. The gender are encoded as follows, female as 2 and male as 1. Multiple experiments are conducted for each subject, each experiment characterized by a specific description, which includes walking speed and inclination. The experiments considered in this study are those where the inclination is zero. Each experiment can be repeated multiple times, with the reproductions noted in the time scale dimensions (n x m, where m represents the number of experiment repetitions, and n the number of points measured per experiment). There are 2 types of time frame, the absolute time (Time in s) and the normalized time (time_norm). For each leg (Right and Left), the respective trajectories are stored under the name of the Axes that represent the type of DoF (X for the flexion axe and Y for the abduction axe).

4.2.2 Key Points Extraction

In order to have discrete outputs for the NN instead of the continuous trajectories, specific key points are extraded from the trajectories. The initial step involved identifying the extremums of the curves for each of the DoFs. As discussed in the literature as [7], these shapes' patterns that can be easily identify are the extremums of functions, with the number of extremums being consistent per DoF. Additionally, these extremums occur at fixed gait percentage intervals.

Precisely identifying the desired extremums is crucial for the subsequent analysis, as there can be more extremums computed than desire. The extremums will be identified using the Python library Scipy, with the *brentq* function. *brentq* takes as parameters a function F and two points a,b and identify the value c between a and b where the function is equal to zero. So to have this function F, the interpolation between the trajectory points from the database is made using the Scipy Spline interpolation function, giving at the end a function. It can then be derivate, giving an output another function that can be used with *brentq*. For each time step, if the function at t and t+1 have different signs, *brentq* is used to find the root between the two time steps.A similar method will be employed to identify inflection points by analyzing the second derivative.

After analysis of the data collected, the number of extremums obtained can vary greatly from what is expected. According to the literature [7, 44], the number of extremums for a DoF is fixed regardless of the test subject and walking speed. However, the method described above shows that the number of extremums detected can vary greatly. This is also linked to the shape of the curves, ranging from trajectories with a more classical shape for DoFs, as in fig. 4.6, to trajectories with more different shapes, fig. 4.7. The aim is to have a NN model with a constant number of outputs per DoF, and therefore a constant number of key points for all DoFs. It is necessary to standardize the number of extremums and inflection points

with the literature.

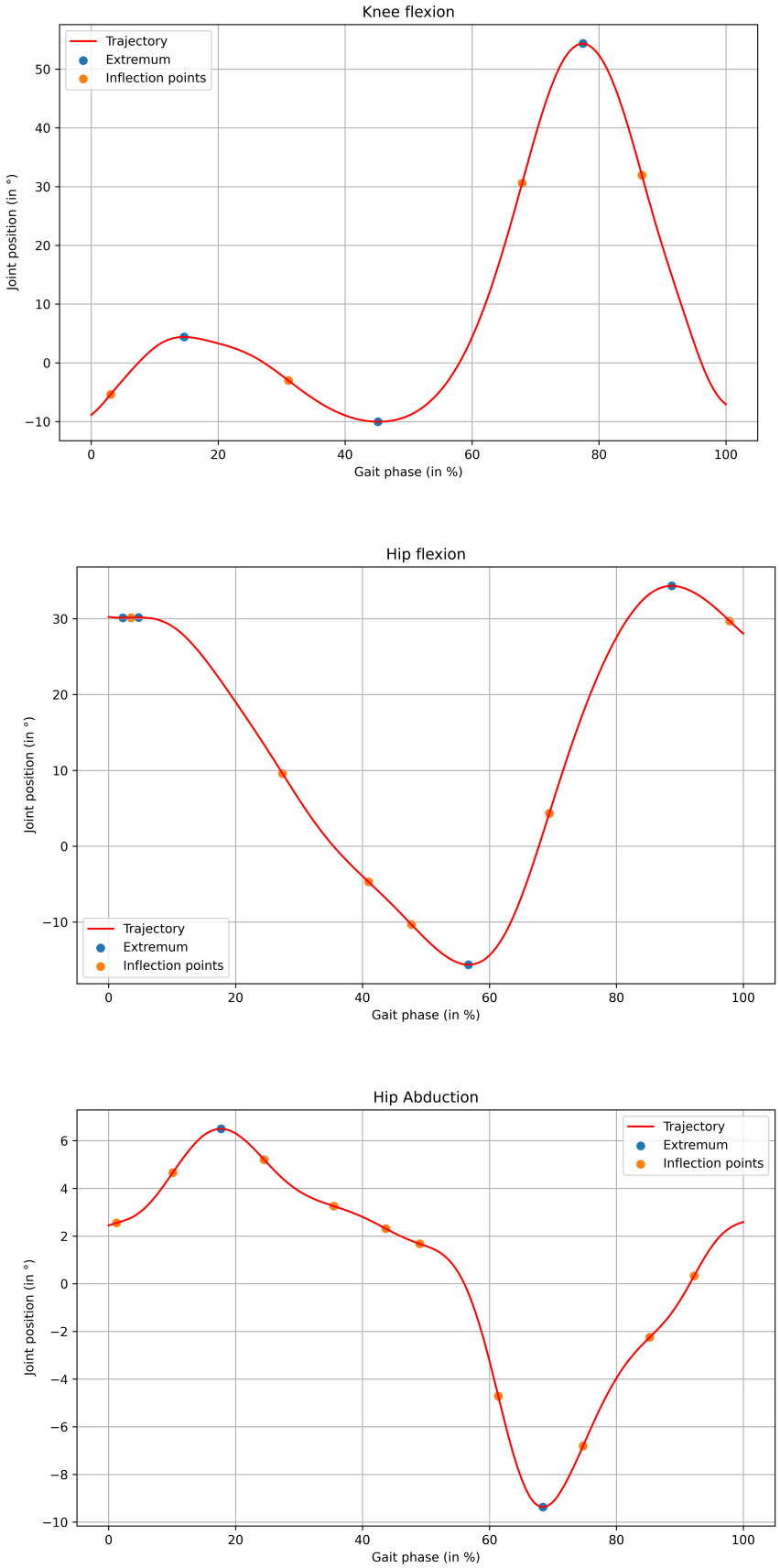


Figure 4.6: Example of a representation for the same subject of trajectories with a more “classic” form of the desired DoFs. Note that for the hip flexion there are already more points than expected to be recurrent in the literature (see section 4.2.2). What’s more, by considering only inflection and extremum points as key points, their number is already very different from one DoF to another, which is contrary to the desired objective of having the same number of key points per DoF.

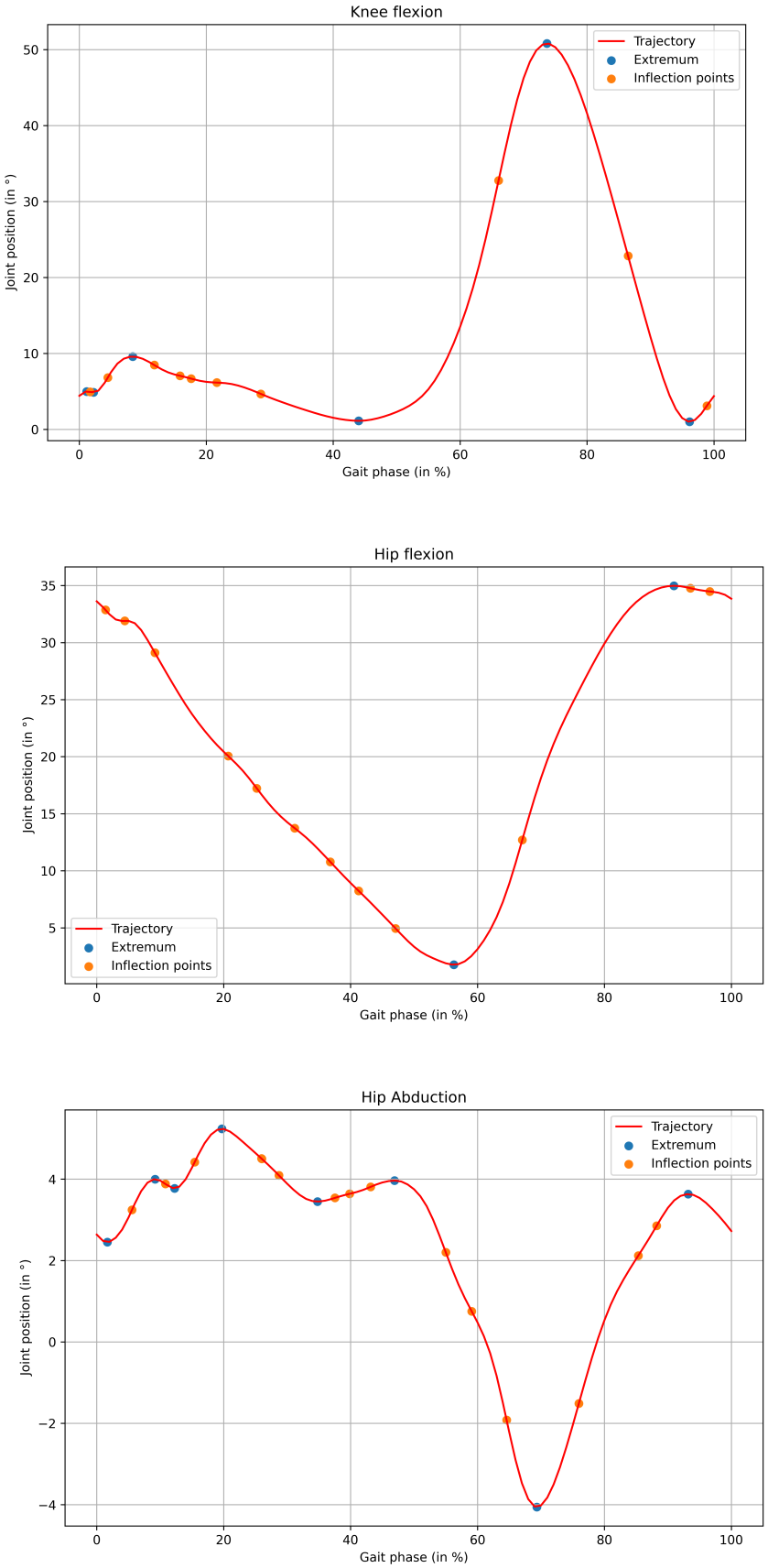


Figure 4.7: Another example of a representation for the same subject of trajectories with a shape further removed from the usual shape of the desired DoFs. It is apparent that the number of extremum and inflection points is more variable than in fig. 4.6, especially in the hip abduction..

In order to standardize the number of extremums and inflection points, while ensuring good curve reconstruction in the end, it is first necessary to be able to identify for each extremum its nature (maximum or minimum) and the gait percentage interval in which to find it. In cases like fig. 4.7, some of the determined extremums align with those being sought. By knowing the nature of the desired extremum (maximum or minimum) and the interval in which it is expected, we can select the one that most closely matches the intended one. To delve deeper, the identification of extrema facilitates data filtering as there are certain trajectories that do not comply with the conditions, they will simply be removed. As demonstrated in fig. 4.7 compared to fig. 4.6, there can be an excessive number of candidate extrema, and conversely, instances where there are no candidates in the desired interval. Consequently, the trajectories selected for training are those where, for each degree of freedom (DoF), at least one candidate point is identified within each interval where an extremum is expected. This approach leads to a minimal data loss of approximately 5%, ensuring that all trajectories used for subsequent training contain the requisite key points.

Similarly, in order to standardize the number of inflection points, which can be seen to vary greatly in fig. 4.6 and fig. 4.7, a rule has been established: an inflection point is only selected if it lies between two previously selected extremums. And if several candidates are possible between two extremums, only one will be selected at the end, depending on the trend of the curve between the two extremums. If the curve is rising, the inflection point selected is the one with the highest value of in its derivative. If the curve is decreasing, the inflection point selected is the one with the smallest value of in its derivative, which can be negative. Physically, the inflection point selected is the one corresponding to the maximum/minimum speed between the two extreme positions.

The key points selection process begins with the calculation of all extrema in the trajectory. These extrema are then classified into the predefined intervals based on their gait percentage. Subsequently, in an interval, the candidate extremum with the highest (or lowest) value is selected, depending on the nature of the extremum required.

A similar methodology is applied to the selection of inflection points. After identifying all potential inflection points, they are categorized into intervals based on their gait percentage, specifically between two consecutive extrema. The selected inflection point is determined by the nature of the trajectory between the two extrema, with the point of maximum or minimum speed being chosen accordingly.

Finally, the selection of additional points varies according to the DoF. The following sections will provide a detailed analysis of the extremum positions, the choice of additional points for each DoF.

Knee flexion key points

Regarding knee flexion, research, such as that presented in [7], identifies three key extrema. Their characteristics and corresponding gait percentage intervals are as follows:

- A maximum at e_{KF1} within the interval [0%, 20%] of gait cycle
- A minimum at e_{KF2} within the interval [25%, 60%] of gait cycle
- A maximum at e_{KF3} within the interval [65%, 90%] of gait cycle

Based on the number of extrema, the rule defined earlier allows for the deduction of two inflection points, which are selected following the aforementioned procedure. Also, the points at gait percentage 0 and 100, qualified as e_{KFA} and e_{KFZ}, are added as boundaries conditions for a better interpolation. The position of the joint at these two points is taken to be the same, to ensure that the trajectories for the same individual at a given walking speed taken

end to end are continuous. Additionally, supplementary points are chosen to enhance the approximation of the curve's shape during interpolation. These points are:

- The point at $\frac{e_{KFA} + e_{KF1}}{2}$. It serves to better interpolate the first part of the curve in the interval $[e_{KFA}, e_{KF1}]$.
- The two points at $\frac{e_{KFi} + e_{KFi+1}}{2}$ with i = 1,2. They serve to add another point with the respective inflection points in the intervals $[e_{KFi}, e_{KFi+1}]$ for a better interpolation.
- Lastly, 2 final points in the interval $[e_{KF3}, e_{KFZ}]$. The first is fixed at gait percentage $\frac{e_{KF3} + e_{KFZ}}{2}$. The second one, at gait percentage $0.2.e_{KF3} + 0.8.e_{KFZ}$, obtained by trial and error, as it has generally been noted that there is an approximation error around this chosen gait phase.

In the end, we obtain 12 key points for knee flexion, with 3 extremum points, 2 inflection points, 2 points as boundary conditions and 5 additional points.

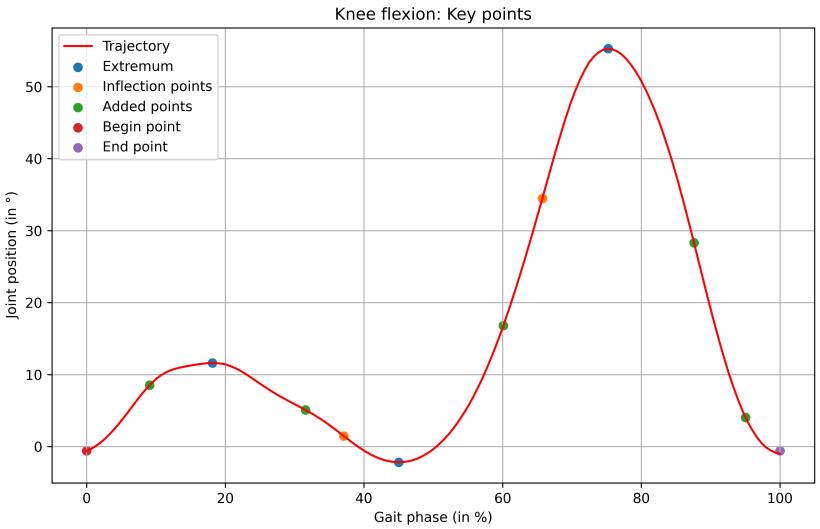


Figure 4.8: Key points representation for the knee flexion.

Hip flexion key points

Regarding hip flexion, the same researches [7], identifies two key extrema. Their characteristics and corresponding gait percentage intervals are as follows:

- A minimum at e_{HF1} within the interval [40%, 65%] of gait cycle
- A maximum at e_{HF2} within the interval [70%, 90%] of gait cycle

Based on the number of extrema, the designed rule allows for the deduction of one inflection points, which are selected following the aforementioned procedure.

When adding the boundaries conditions with the e_{HFA} at gait percentage null and e_{HFZ} at gait percentage 100 put at the same joint position, there are seven additional points to be included. In this case, as illustrated in fig. 4.9, the points were selected through extensive trial and error. The regions with the highest concentration of additional points correspond to areas where the greatest variation in the data was observed. This is particularly evident near the end of the gait cycle, where a higher density of points, relative to their gait percentage distance, is present. The final additional points are:

- The 3 points in the interval $[e_{HFA}, e_{HF1}]$ which are in order: $\frac{9.e_{HFA}+e_{HF1}}{10}$, $\frac{e_{HFA}+e_{HF1}}{2}$, $\frac{e_{HFA}+3.e_{HF1}}{4}$.
- The 3 points in the interval $[e_{HF1}, e_{HF2}]$ which are in order: $\frac{9.e_{HF1}+e_{HF2}}{10}$, $\frac{e_{HF1}+3.e_{HF2}}{4}$, $\frac{e_{HF1}+9.e_{HF2}}{10}$.
- One point in the interval $[e_{HF2}, e_{HFZ}]$ which is: $\frac{e_{HF1}+e_{HF2}}{2}$.

In the end, we obtain 12 key points for hip flexion, with 2 extremum points, 1 inflection points, 2 points as boundary conditions and 7 additional points.

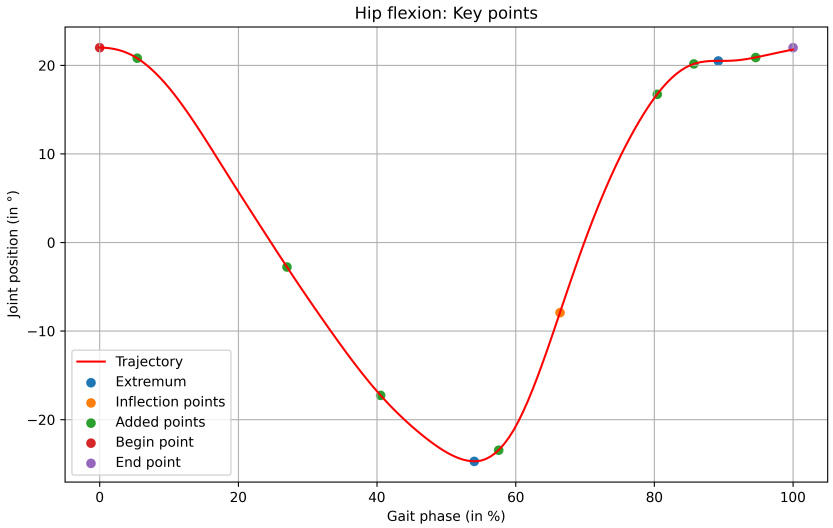


Figure 4.9: Key points representation for the hip flexion.

Hip abduction key points

Regarding hip abduction, the same researches [7], identifies two key extrema. Their characteristics and corresponding gait percentage intervals are as follows:

- A maximum at e_{HA1} within the interval [5%, 30%] of gait cycle
- A minimum at e_{HA2} within the interval [55%, 80%] of gait cycle

Based on the number of extrema, the designed rule allows for the deduction of one inflection points, which are selected following the aforementioned procedure.

Similarly to the hip flexion case, additional points were concentrated in areas exhibiting the greatest interpolation errors. As shown in fig. 4.10, there is a higher density of additional points following the first extremum. This is primarily because significant data variations were detected in this region. In fact, some of these variations could be mathematically evaluated as extrema. However, it is important to note that this particular curve is not present in all trajectories. After adding the e_{HAA} and e_{HAZ}, the final additional points are:

- One point in the interval $[e_{HAA}, e_{HF1}]$ which is: $\frac{e_{HAA}+e_{HF1}}{2}$.
- The 4 points in the interval $[e_{HF1}, e_{HF2}]$ which are in order: $\frac{e_{HF1}+e_{HF2}}{2}$, $\frac{3.e_{HF1}+2.e_{HF2}}{5}$, $\frac{3.e_{HF1}+e_{HF2}}{4}$, $\frac{2.e_{HF1}+3.e_{HF2}}{5}$.
- Two points in the interval $[e_{HF2}, e_{HAZ}]$ which are in order: $\frac{e_{HF2}+end\_point}{2}$, $\frac{e_{HF2}+4.e_{HAZ}}{5}$.

In the end, we obtain 12 key points for hip abduction, with 2 extremum points, 1 inflection points, 2 points as boundary conditions and 7 additional points.

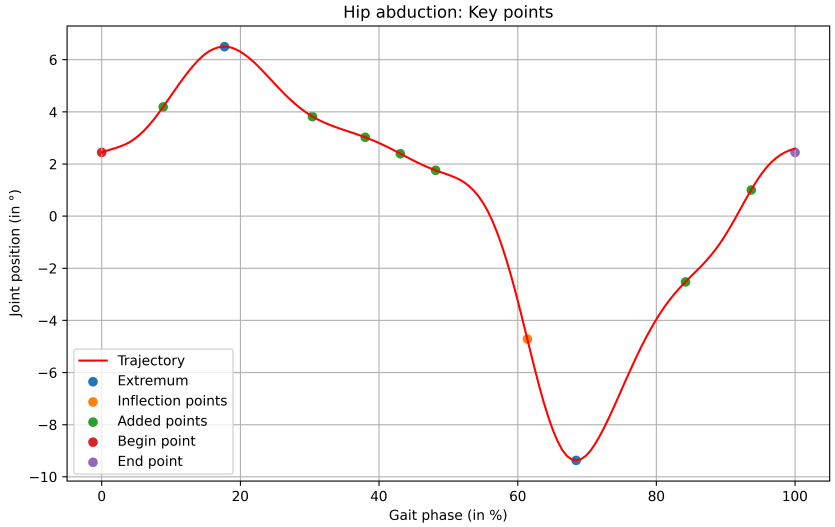


Figure 4.10: Key points representation for the hip abduction.

4.2.3 Neural Networks Training

The training mode of the neural network model can be subdivided into several steps: data preparation and training set-up.

Data Preparation

First, there is data preparation. Each trajectory is linked to a subject and their details.

- An initial function *extract_data_to_train* takes as parameters the ordered database in the file represented by fig. 4.5, and for one DoF, the details of the subjects to be extracted, the conditions on these details (for example, only subjects with an age below a maximum), and the number N of times a trajectory is repeated. This function ultimately creates 3 matrices. One matrix has each of its rows representing a trajectory. One other has each of its rows representing the normalized time scale of the corresponding trajectory. A trajectory and its normalized time scale are repeated N times in each matrix. The third matrix is the input data matrix, where each row represents the biomechanical parameters of the subject that were requested as input, the walking speed corresponding to the trajectory. Additionally, a pair (gait phase, joint position) is randomly chosen from the points of the trajectory provided in the database and added to the corresponding row of inputs. This pair is different for each repetition of the trajectory. This allows for multiplying the amount of data used to train the neural network.

- The second function *compute_data_for_training* encapsulates the first and allows transforming the two tables with the trajectories and their time scales into a table containing, for each row, the key points corresponding to the trajectory.

Thus, in the case of a complete model as shown in fig. 4.3, a sample for a trajectory corresponds to the inputs: age, height, weight, gender, walking speed, and the pair (gait phase,

joint position) randomly chosen from the trajectory, and the outputs are the key points of this trajectory. This allows for a base of 5142 samples for N=1, meaning each trajectory is only used once. This can be multiplied by increasing the value of N.

After extraction and calculation of data via *compute_data_for_training*, the input data will be standardized using the StandardScaler from the Python library Scikit-learn [57] under sklearn.preprocessing to benefit from the advantages described in section 2.4.1. These extracted data will be divided into two parts using the `train_test_split` function from the Scikit-learn sub-library `sklearn.model_selection`. The split is 0.8 for training and 0.2 for test. One part will be used during training for training and validation, and the other part will only be used at the very end to test the model's performance.

Training Set-up

The second step involves the model itself. A function has been created to design the skeleton of the neural network (NN). This function, *create_model*, takes the following parameters:

- the shape of the input input_shape
- the learning rate of the optimization function to be used
- the activation function to be used in the model
- the number of neurons in the hidden layers
- the dropout value dropout_rate (see section 2.4.3) between the network layers
- the regularization function to be used in the network
- the method for initializing the weights of the neurons (see section 2.4.3) initializer

This function particularly utilizes the Keras sub-library of TensorFlow [64]. The *create_model* function operates in several stages:

- First, the initializer for the weights is identified by its name, and it needs to be assigned to the corresponding weight initialization method. For each coded name, a corresponding initializer object from the Python sub-library `tensorflow.keras.initializers` is assigned (e.g., `HeNormal`, `GlorotUniform`).
- The same applies to activation functions (see section 2.4.3). Although the ReLU and ELU functions are known by the names `relu` and `elu` in TensorFlow libraries, to use other functions, their names need to be encoded (e.g., `LeakyReLU` as `leakyrelu` and `PReLU` as `prelu`). After being passed into the function, if the name is known, it will be used by the Python objects. Otherwise, for `LeakyReLU` and `PReLU`, they must be extracted from the Python sub-library `tensorflow.keras.layers`, and the encoded names replaced with the corresponding objects before being provided to the rest of the TensorFlow objects.
- The creation of the model begins here. It is initialized with the `Sequential` object from the `tensorflow.keras.models` library to specify that it will be a neural network. Then, the various layers are added. Being dense layers, the `Dense` object from the `tensorflow.keras.layers` library is used. For the first layer, the shape of the input is specified, along with the activation function, weight initialization method, and regularization function used (None if there isn't one). Here, 3-7 hidden layers will be used (this may evolve during the training process). If the `dropout_rate` is greater than zero, a `Dropout` layer from the `tensorflow.keras.layers` library is added between

each layer of the neural network. Finally, the output layer with 24 outputs (12 key points * 2) and a linear activation function is specified.

- The activation method used here is the Adam optimizer (see section 2.4.3), with the learning rate specified as an input. Finally, the model is compiled with `MSE` for loss, `MAE` for another metric, and the previously defined activation method, then return by the function. The two metrics can be seen in table 2.2.

The idea behind the neural network is to establish multiple regressions between the input features and the key points. This function was created because, for proper functionality of the program, it will be wrapped by a `KerasRegressor` object from the Python `scikeras` library, which requires a function that returns a model as output. However, the shape of the input data, which is given as a parameter to the `create_model` function, must be specified before wrapping the function with `KerasRegressor`. For this purpose, the `partial` function from the Python `functools` library is used, allowing the creation of a new function from an existing one where one or more parameters of the original function have already been specified. Thus, a new function `create_model_with_input_shape` is obtained, where the input shape is already specified.

For training the model, the method used will be a GridSearch via the `GridSearchCV` object from the Python library Scikit-learn [57]. The `KerasRegressor` object ensures compatibility between the neural network created using the TensorFlow library and the `GridSearchCV` from the Scikit-learn library.

GridSearch is a process that involves training a Machine Learning model using different combinations of hyperparameters. For each model, various hyperparameters are tested with a range of values. Subsequently, each model, configured with a unique combination of hyperparameters, undergoes training using a cross-validation method.

Cross-validation is a robust method for evaluating the performance of a machine learning model by partitioning the dataset into subsets. The dataset is typically divided into k equal-sized folds. The model is trained on k-1 folds and validated on the remaining fold. This process is repeated k times, with each fold serving as the validation set once. The final performance metric is obtained by averaging the results from all k iterations, providing a comprehensive assessment of the model's generalization capability.

The parameters that will be varied during the GridSearch include the learning rate for the Adam optimizer, the number of neurons per hidden layer, the activation function, the weight initialization method for each neuron, the batch size, the number of training epochs, the dropout rate (including 0 for no dropout), and the regularization functions (including an option for no regularization). The final objective is to identify the optimal combination of values for these parameters.

Table 4.1: Hyperparameters and their choice of values

Hyperparameters	Choice of values
Batch size	30, 50, 60
Epochs	300, 400, 450
Learning rate	0.001, 0.004, 0.01
Activation function	relu, elu, leakyrelu, prelu
Number of neurons	64, 128, 512
Dropout rate	0.0, 0.2, 0.5
Weight initialization	HeNormal, GlorotUniform
Regularization	None, L1, L2

At the conclusion of the GridSearch process, the model exhibiting the highest performance

is selected. The performance metric utilized for all models is the Mean Squared Error (MSE) as shown in table 2.2, which serves as the loss function.

4.2.4 Gaussian Process Regression Training

As seen in section 2.4.2, a Gaussian Process Regression Model consists of two parts: the constant kernel and the variable part of the kernel. This variable part can have several natures, and the best nature for a specific case must be determined with its corresponding parameters.

The Gaussian Process Regression is used in two cases: the estimation of the gait phase and the estimation of the gait period. The variation will depend on the data used.

- The `compute_data_gait_period_training` function extracts subject details (age, height, weight, gender) and walking speed from the database to be used as input, and the gait period to be used as output. The complexity arises from the fact that the data from [17] are normalized with respect to time and do not provide this information. Therefore, it is necessary to recognize in the database which time scales have been normalized or not, hence the difference in the database between `time` and `time_norm` in the organization of file fig. 4.5. This results in 4394 samples compared to the previous 5000+ by removing the data from [17].

- The `compute_data_intent_recognition` function extracts the data for estimating the gait phase. For a given subject and experiment, all trajectories for one leg will have the same time scale. In this case, we consider the trajectories (with at least a hundred points) of the 3 desired DoFs and their normalized time scale (expressed as a gait percentage). Interpolations are applied to these trajectories via the `CubicSpline` function from the Python library Scipy [58]. This allows for easy calculation of the derivative of these trajectories and obtaining the velocity. From there, a gait phase value is selected (one from the initial normalized time scale to obtain the real values for the positions), and the position and velocity values of the different DoFs at the chosen gait phase are selected. Thus, the output is a gait phase and the input consists of the various positions and velocities of the related DoFs. This process can be repeated several times for the same trajectory. If this process is done 10 times, it gives 51420 samples.

Once the data is obtained, the input data is standardized using `StandardScaler` from the Python library Scikit-learn [57]. From here, the training process is the same in both cases. It is done using `GridSearchCV` from the Python library Scikit-learn. Different possible kernels will be used as grid parameters. These kernels are written in the form: `C(1.0, (1e-4, 1e2)) * VariableKernel(parameters_bounds=(1e-6, 1e2))`. `C` is the abbreviation for `ConstantKernel`, an object from the Python sub-library `sklearn.gaussian_process.kernels` of Scikit-learn, representing the constant part of the kernel of the Gaussian Process Regression. The first value is the initial value of the `ConstantKernel`, and the pair `(1e-4, 1e2)` represents the limits within which the value of this constant part will be searched. For the variable part of the kernel, there are several types, all from the Python library Scikit-learn. The types of kernels considered include the Radial Basis Function (RBF) kernel, Matern, RationalQuadratic, ExpSineSquared, and DotProduct. They all have a parameter specifying the bounds within which the `length_scale` will be searched, and they have their own specificities. For Matern, there is the ν parameter whose value must be specified, often chosen between 0.5, 1.5, and 2.5 (see Section X). For RationalQuadratic, the limits for searching the α value must be specified, here $(1e-4, 1e2)$. For ExpSineSquared, the limits for searching the

periodicity must be specified, here $(1e-2, 1e2)$. From there, GridSearch will determine the best parameters for each kernel's variable and constant parts and determine the best kernel. Also, during training, if a parameter value is too close to the given limits, the program will issue a warning and it can be adjusted.

4.2.5 Interpolation Choices

As shown in fig. 4.1, the key points will be interpolated to reconstruct the trajectory. The work of Moissenet [44] shows that a spline interpolation of degree 5 with fewer key points than those used here gives good results. Here, based on Moissenet's results, a spline interpolation of degree 3 using the Python library Scipy gave better results in our case as in section 5.3.

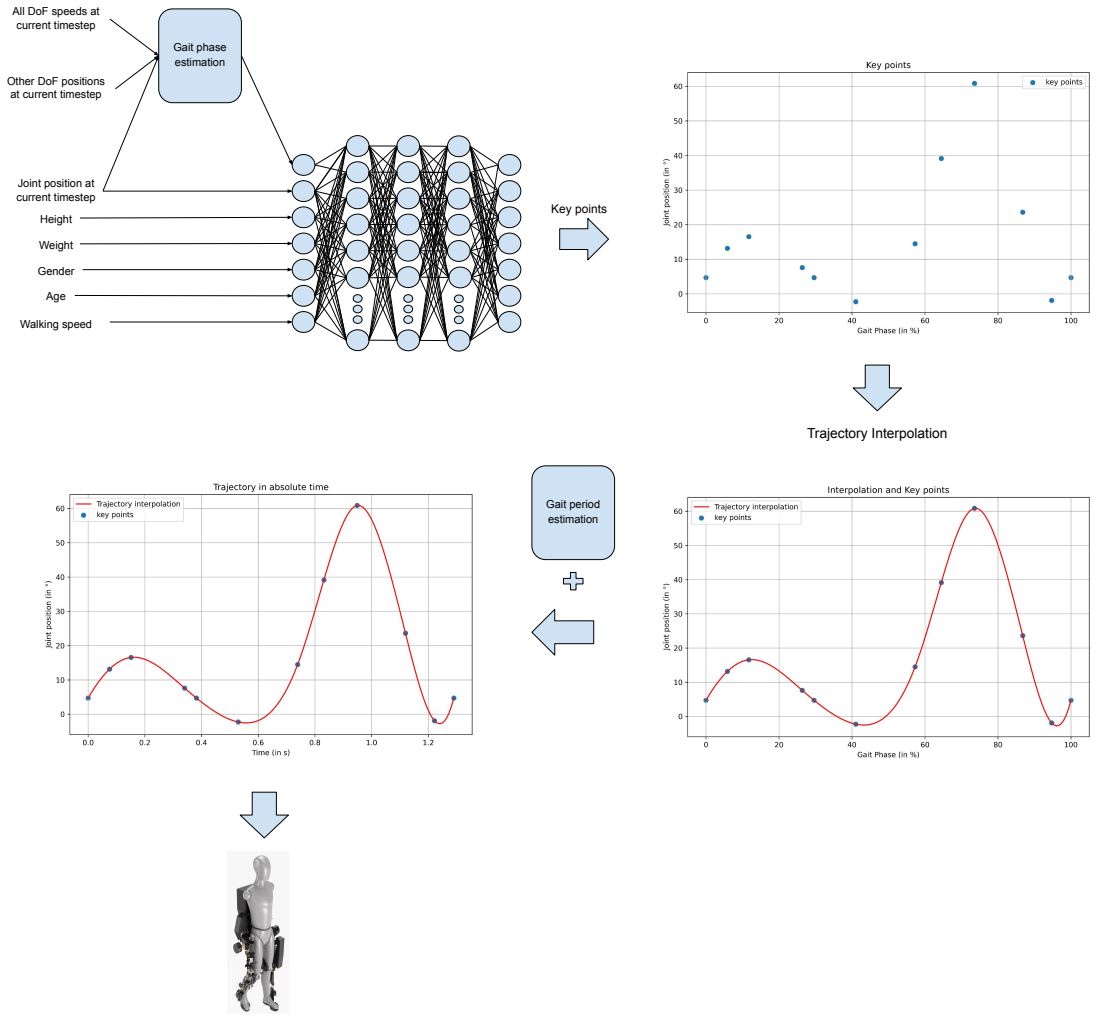


Figure 4.1: Initial concept for a high-level machine learning controller for a single degree of freedom (DoF). It is structured into two primary components: Gait phase estimation and Trajectory Generation. The Gait phase estimation component identifies the current gait phase, determining the percentage of the gait cycle at which the system is operating. Trajectory Generation involves a two-step process. First, a neural network processes inputs that include pilot-specific parameters (age, weight, height, gender), walking speed, the current position of the DoF, and the identified gait phase. These parameters ensure the continuity of the generated trajectory with the current position. The neural network then outputs key points, which are subsequently interpolated to create the desired DoF trajectory. It is then expressed in terms of absolute time via the Gait period that is estimate and then is provided to the rest of the system.

Chapter 5

Performance Evaluation

This chapter evaluates the controller's performance. As the controller can be divided into several sub-parts, the performance of each sub-part will first be analyzed independently, i.e. gait phase estimation, gait period estimation, interpolation performance and neural network performance. Finally, the various elements will be put together to evaluate the controller's performance.

5.1 Gait phase Estimation Performance

For the model estimating the gait phase shown in Figure X, the results are not convincing. Indeed, the obtained model is:

- ConstantKernel: 31.6**2
- Nature of variable kernel: RationalQuadratic
- Parameters of the variable kernel: alpha=0.0198, length_scale=0.019

The obtained MSE is 125.20, which corresponds to an error of 11.18 on the approximated gait phase. Training a GPR model is very resource-intensive, and beyond a certain number of samples, specifically 51,420, it is necessary to compress the data matrix to use it within the RAM memory of a computer.

The error obtained is due to high error at the boundaries. Since gait is a cyclical process, being at 0% gait and 100% gait is similar. Seyrath and Barnikol [1] propose a method to resolve this issue, but it will not be applied here and can be considered a potential improvement.

5.2 Gait Period Estimation Performance

As seen in section 4.2.4, the Gaussian Process Regression model for gait period estimation is trained using a GridSearch. The operation of Gaussian PRocess Regression was explained in section 2.4.2. GridSearch designates, for each type of variable kernel, an initial kernel whose hyperparameters are updated during training to determine the optimal shape of this kernel. This optimal shape is sought within the hyperparameter intervals as described in section 4.2.4. At the end, it evaluates which of these initial kernels has achieved an optimal shape with the best performance. Performance is evaluated using the MSE, which can then be evaluated more concretely by taking its square root, since the output of the model is an

estimate of the gait period, and the root of the MSE allows us to evaluate the error in seconds made on this estimate.

In table 5.2 are shown the different kernels obtained, ranked from worst to best by Gridsearch and the root of the MSE results with Training Data. GridSearchCV from the Python library Scikit-learn only provides the starting points before updating, and only the best kernel is provided with its updated hyperparameters. This shows that the best kernel is contested by the two in green. Sometimes, if you run GridSearch several times, the best kernels are interchangeable between these two. This is understandable, given the fact that both the Matern and the RationalQuadratic kernels are flexible enough and able to capture many different patterns. Both kernels have some mechanisms to model different smoothness and variability, and this could be another reason whereby performance metrics can turn out quite similar on an identical data set. If this data contains properties that are on the opposite ends of the spectrum of characteristics for which both kernels perform well, such as smooth transitions and quite different scale features, then the MSE performance metrics will be close to each other for both models.

At the end of this training, for the case of the Matern kernel where the corresponding model with the updated values are:

- Constant Kernel: 3.63^2
- Kernel Type: Matern
- Parameters: length_scale=8.5, nu=1.5

A representation of the model can be seen in fig. 5.1. Walking speed was chosen here because it correlates most closely with gait period. Other factors such as age are not sufficient to explain the gait period on their own, so when age is plotted against gait period no real correlation can be deduced. In fig. 5.1, it can be observed that the training or test data fall within the 95% confidence interval of the model, except at very low speeds. One hypothesis could be that the input parameters (height, age, weight, gender, and walking speed) are insufficient to estimate the gait period at these low speeds. However, given that the overall root of the MSE using the Test data is 0.054 s. Relative to typical gait period values [0.5, 4] seconds, the error is in the order of 1.35% to 10%.

Table 5.1: Kernel Performance Evaluation with Training Data from worst to best. The GridSearch look for each type of variable kernel, the best hyperparameters to approximate the model and then compare the different models obtain for each type of varaible kernel. The worst kernels are in red and the best are in green.

Constant Kernel	Type of Variable Kernel	Parameter of Variable Kernel	Root of the MSE
1^2	ExpSineSquared	length_scale=1, periodicity=1	0.1989
1^2	RBF	length_scale=1	0.0874
1^2	RationalQuadratic	alpha=1, length_scale=1	0.0574
1^2	Matern	length_scale=1, nu=1.5	0.0571

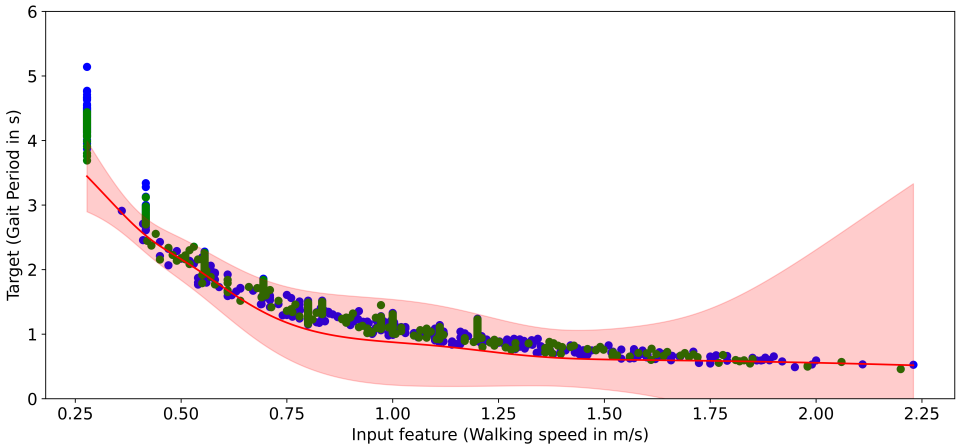


Figure 5.1: Representation of gait period estimation model using Gaussian Process Regression. The model takes several features as input, the one represented here is the walking speed on the x-axis and the predicted gait period on the y-axis. The red line corresponds to the predicted value, and the red range corresponds to the standard deviation of the result. More precisely, the red range represents the 95% confidence interval, i.e. the model assures that 95% of gait periods will fall within this red range. This is confirmed by the fact that the blue dots represent the data used for training the model and the green dots are used for testing. These points are well within the model's confidence interval.

5.3 Evaluation of Trajectory Reconstruction Accuracy Using Selected Key Points and Interpolation Method

Here, the aim is to evaluate the error made when recreating a curve from selected key points and an interpolation method. As seen in section 4.2.5, the spline interpolation was chosen on the basis of Moissenet et al [44]. In their case, a spline interpolation of degree 5 was chosen. The spline interpolation method used here is performed using *make_interp_spline* from the Scipy python library. The output is a function that can then be evaluated at certain points. Knowing the standardized version of the curve as a function of gait percentage, we can evaluate the interpolated curve at the same gait percentages as the real curve. We then calculate the point-by-point RMSE between the interpolated curve and the actual curve, and average these points to obtain the RMSE between the two curves.

However, an RMSE of 2° between the two curves does not represent the same thing if the curve has an amplitude of 10° or 60°. Thus, the decision was made to divide this error by the amplitude of the real curve (the difference between the maximum and minimum of the curve) to precisely determine if this error is significant or not giving what will be called a relative RMSE. This calculation can be used to determine the best degree of spline interpolation to use here. The results of this comparison can be seen in table 5.2. It's clear from this table that, thanks to the Python library used here for spline interpolation, particularly in the case of hip flexion, the best interpolation degree is 3.

In order to graphically observe the highest (worst-case) values that this relative RMSE can take for a spline interpolation of degree 3, fig. 5.2 has been established. The choice of representation as a function of gait speed is arbitrary. It shows that in the vast majority of cases, the relative RMSE is less than 6% of the curve amplitude, and in the worst case less than 10%.

In fig. 5.3, we observe the best interpolation cases with errors below 1%. However, in the case of fig. 5.4, there are instances where not only the relative RMSE but even fig. 5.2, with

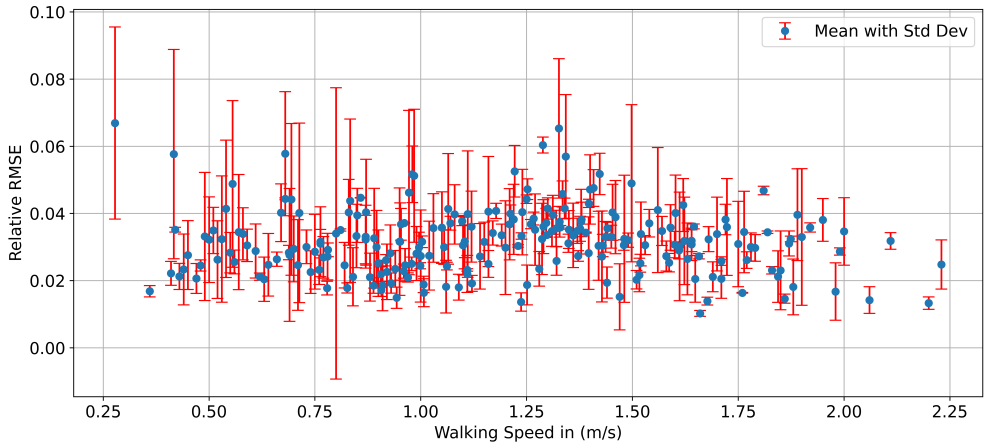
the mean and standard deviation, fail to visualize them—cases where the relative RMSE is greater than 10%. After calculation, these data represent 0.1% of the data for knee flexion, 0.3% of the data for hip flexion, and 2% of the data for hip abduction. The proportion of these cases is low and does not invalidate the choice of this interpolation method. However, these data should have been excluded for training the neural networks, retaining only key point cases where the interpolation error is 10% or less.

Table 5.2: Relative RMSE by Spline interpolation Degree for Knee Flexion, Hip Flexion, and Hip Abduction. There are the mean and standard deviation of the relative RMSE between the real curve and the interpolated curve (RMSE / Amplitude of the real curve) for different spline interpolation degrees for each DoF.

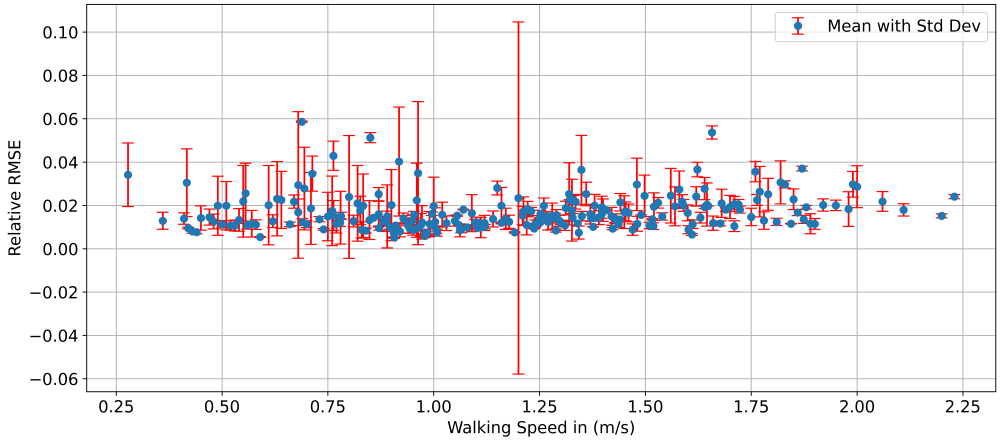
Knee Flexion		
Interpolation Degree	**Mean RMSE**	**Std Dev**
3	0.01762134	0.01162912
5	0.02126864	0.02200282
7	0.03159490	0.04811942

Hip Flexion		
Interpolation Degree	**Mean RMSE**	**Std Dev**
3	0.02144935	0.03900513
5	79.38046069	5678.69244902
7	673.24119680	48180.03101227

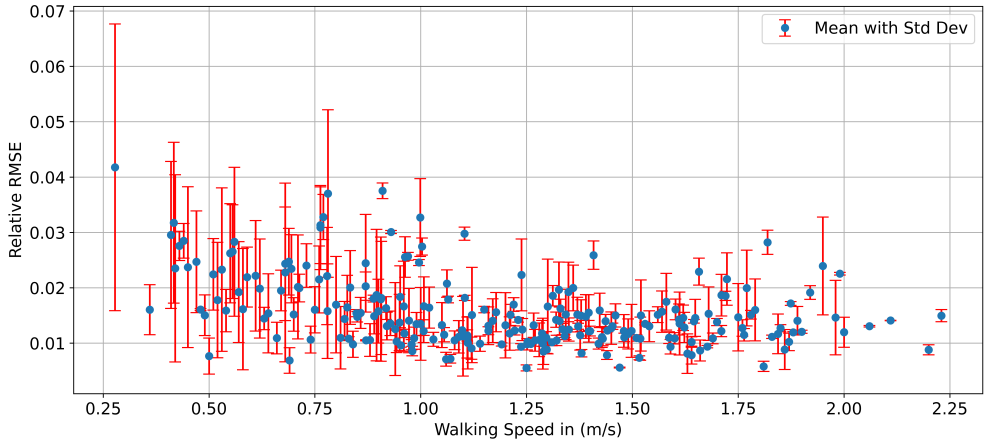
Hip Abduction		
Interpolation Degree	**Mean RMSE**	**Std Dev**
3	0.03678945	0.02643741
5	0.04999707	0.07395576
7	0.15272939	1.14752617



(a) relative RMSE of the interpolation for the Hip abduction relative to the amplitude

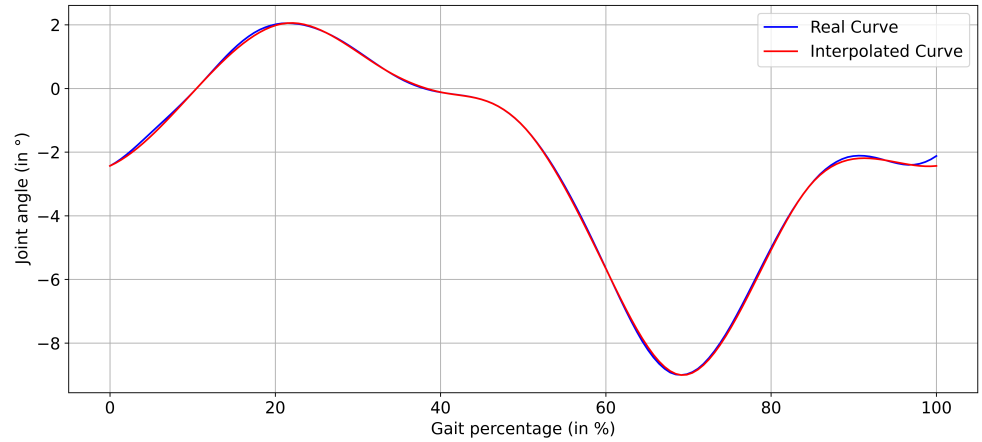


(b) relative RMSE of the interpolation for the Hip flexion relative to the amplitude

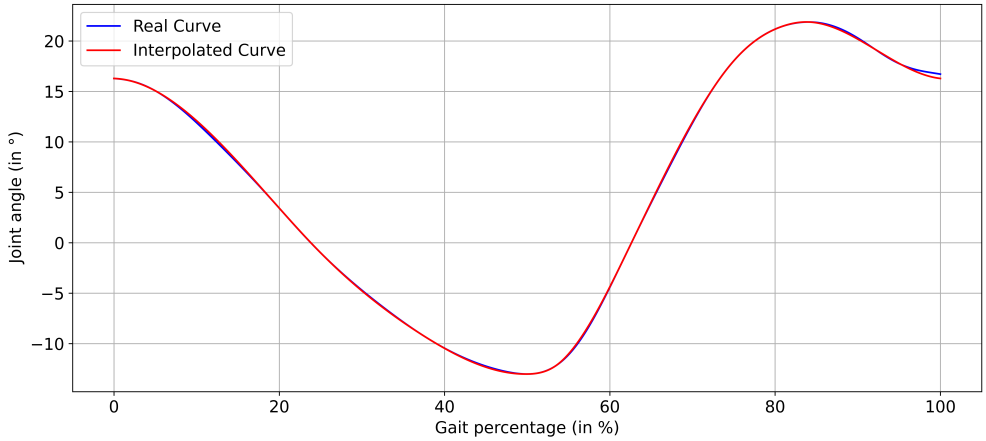


(c) relative RMSE of the interpolation for the Knee flexion relative to the amplitude

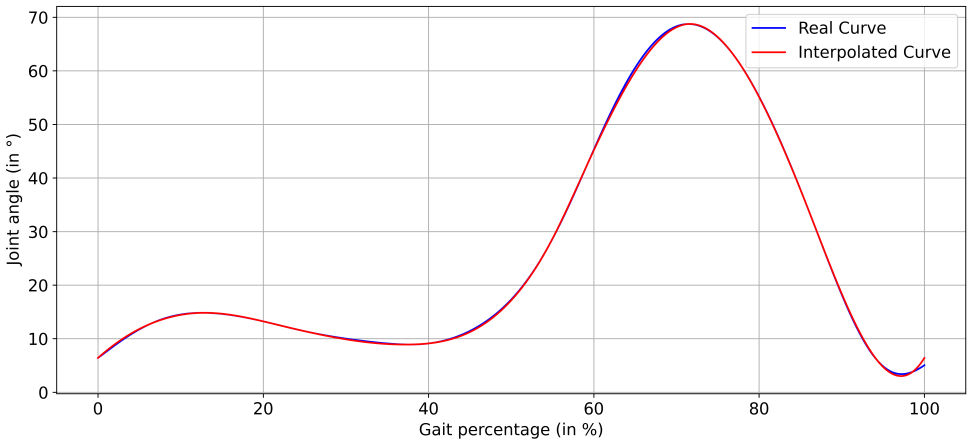
Figure 5.2: Interpolation error relative to the amplitude representation. For the various joints, the factor considered here for the x axis is the walking speed. The choice of Walking speed here is arbitrary, the real objective being to observe graphically which are the worst cases of error, which cannot be observed by taking only the mean of the RMSE. For each running speed, we calculate the mean and standard deviation of the error between the actual curve and the interpolation obtained from these key points, divided by the amplitude of the actual curve (i.e. the difference between the maximum and minimum of the actual curve). For each of the joints, except in special cases, the error between the interpolation and the actual curve represents less than 6% of the amplitude of the actual curve.



(a) Best Interpolation case for the hip abduction

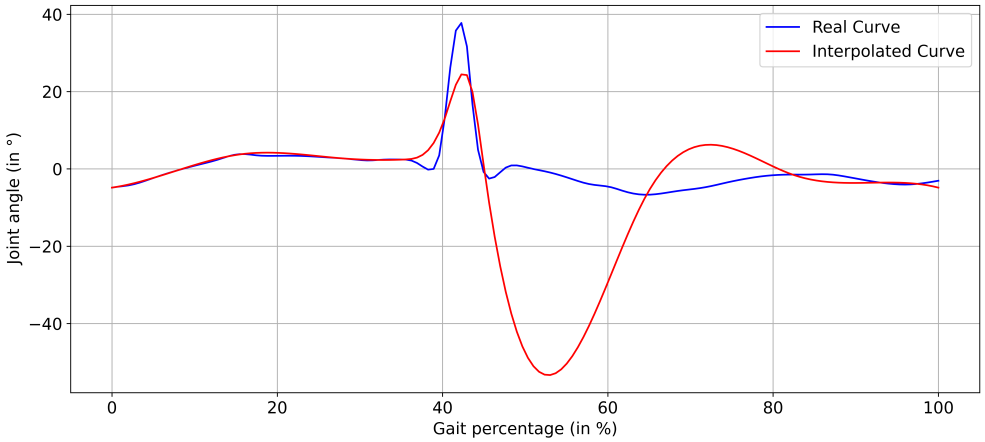


(b) Best Interpolation case for the hip flexion

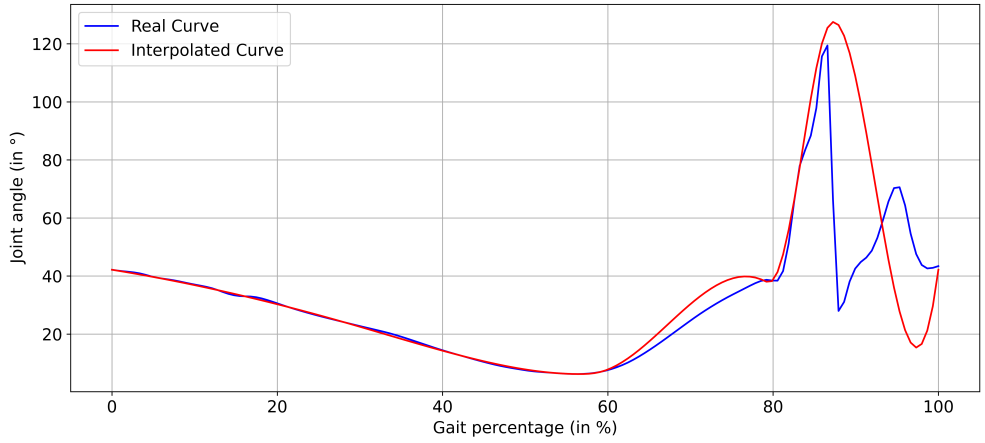


(c) Best Interpolation case for the knee flexion

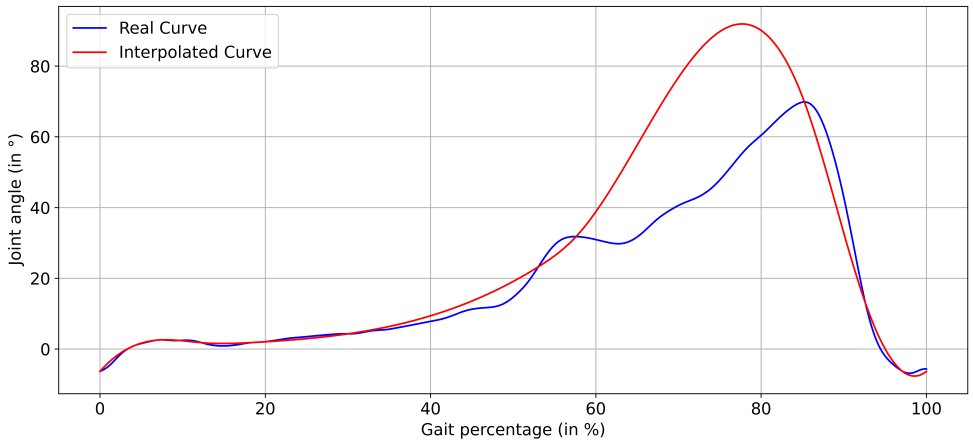
Figure 5.3: Representation of the best interpolation cases for the three DoFs with a relative error of 0.33% for the knee flexion, 0.29% for the hip flexion and 0.52% for the hip abduction.



(a) Bad interpolation case for the hip abduction



(b) Bad Interpolation case for the hip flexion



(c) Bad interpolation case for the knee flexion

Figure 5.4: Representation of bad interpolation cases for the three DoFs with a relative error of 20% for the knee flexion, 15% for the hip flexion and 37% for the hip abduction.

5.4 Neural Networks Performance

The training of the neural network part consisted of two phases. The first phase was carried out with a limited population, in this case male subjects under the age of 25. Also, the neural network used is a reduced version of the basic one, shown in fig. 5.5, where unlike fig. 4.3, age and gender are not considered as input parameters. In a second step, the desired model shown in fig. 4.3 will be trained. In each case, this training is carried out using the GridSearch method.

5.4.1 Error Metrics

The metrics used will be, as a Loss function, the MSE and also the MAE as another function to assess accuracy in the event of regression, as in this case. The R^2 metric is also used.

Mean Squared Error (MSE)

Mean Squared Error (MSE) is the average of the squares of the differences between predicted and actual outcomes. It is computed according to the formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.1}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

Mean Absolute Error (MAE)

The Mean Absolute Error produces the average absolute differences between the observed actual and predicted outcomes. This is calculated by the following formula:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{5.2}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of observations.

Coefficient of Determination (R^2)

The coefficient of determination, usually denoted by the term R^2, is a statistical measure that describes how much variation in a dependent variable is explained as a function of the independent variables in the case of Neural Networks. It indicates how well the neural network model is capable of capturing the underlying trends of the data [65]. The R^2 value ranges from $-\infty$ to 1, where a value closer to 1 indicates a better fit of the model. It gave a note on how much the model explains the variability of the data. R^2 comes by the following formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.3}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value by the neural network, $\bar{y}$ is the mean of the actual values, and n is the number of observations.

5.4.2 Reduced Neural Networks

The aim of the reduced model was to test the performance of the neural network idea with a small population, where gait profiles vary little between individuals, before expanding to populations where gait profile variations are more marked. Thus, in the case of the reduced model, the population was limited to males under the age of 25. Age and gender were removed from the reduced model, as in fig. 5.5, since gender is fixed here and for a fairly young healthy population (19 - 24), age is not a factor that particularly influences gait. This population reduction gives a sample number in our case of 529, multiplied by a factor N (will be 90 for the rest of this phase) as seen in section 4.2.3.

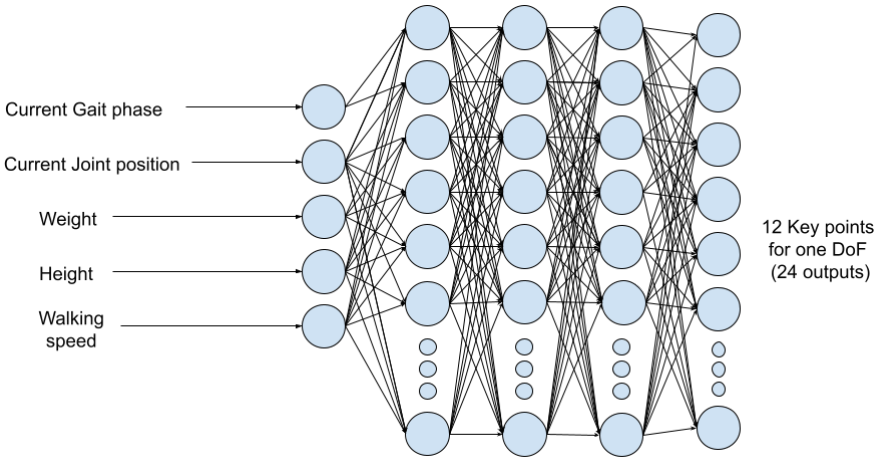


Figure 5.5: Representation of the version of the NN used in the case of the limited population (male under 25). The inputs in contrary to fig. 4.3 don't have the age and gender parameters.

The training process for the models was as follows: a GridSearch was conducted for the model of one DoF, specifically the knee flexion, and the best combination of hyperparameters in this case was used for the models of the other DoFs. During the trials conducted with the reduced model, an observation was made. For the knee flexion model, when using gait percentage values ranging from [0, 100], the results converged at the end of the training with a final loss of 3.68. However, when using the normalized gait phase with values ranging from [0, 1], the best combination of hyperparameters resulted in a model with a reduced size (fewer neurons per hidden layer and fewer epochs required for training), as can be observed in table 5.3. Additionally, the loss results when using the normalized gait phase [0, 1] converged to lower values (2.7 after training), but this is justified by the MSE formula eq. (5.1). An error made on values of [0, 100] will necessarily be reduced if these same values are scaled down to [0, 1]. However, with the normalized gait phase, the memory cost of the model and the training time are greatly reduced.

The decision was made to use normalized gait for the rest of the training and for the other DoFs.

Based on the best combination of hyperparameters for knee flexion from table 5.4, the effect of hyperparameters on the model can be studied. The first case in table 5.4 is when the learning rate is varied. The values of MSE, MAE, and R^2 are obtained with the test data. It can be observed that these values are worse than those of the best model, with the R^2 value in particular indicating that the obtained model does not capture the hidden links in the data. The most significant observation is made with the loss profiles. The loss profile with a learning rate of 0.01 in fig. 5.7 does not converge and shows peaks during its evolution. This is due to a phenomenon visible in fig. 5.6. The learning rate represents the "step" of

Table 5.3: Comparison of Models with Gait Percentage and Normalized Gait Phase. There's a big reduction, in particular a 8-fold reduction in the number of neurons per hidden layer used and a reduction in the number of epochs needed, which overall leads to a shorter training time and a lower memory cost for the model with normalized gait phase.

Hyperparameters	Model with Gait Percentage	Model with Normalized Gait Phase
Number of hidden layers	4	4
Batch size	60	40
Epochs	500	150
Neurons per hidden layer	256	32
Learning rate	0.001	0.001
Dropout rate	0.0	0.0
Activation function	LeakyReLU	LeakyReLU
Regularization function	None	None
Initialization	He normal	He normal
Training time	893.37 s	416.55 s

weight reduction in the neural network, and if it is too large, it can miss the loss minimum and jump to a point where the weight values result in a higher loss, which explains the peaks in fig. 5.7.

Table 5.4: Comparison of Models' performances on test data with same hyperparameters except the learning rate. In every case, the model with the learning rate of 0.001 performs better. And in the case of the model with a learning rate of 0.01, the negative R^2 shows that the model captures no underlying trends in the data.

Error Metrics	Model with 0.001 learning rate	Model with 0.01 learning rate
Test MSE	2.81	3.36
Test MAE	0.85	0.97
Test R^2	0.48	-0.39

It is also possible to vary the number of neurons per hidden layer, starting from the basic model with normalized gait phase. The results are shown in table 5.5. Although the model with 128 neurons per hidden layer performs slightly better in MAE and MSE, the R^2 is lower. This may be due to a lack of data. The model is more complex, but the amount of data doesn't allow the model to capture the underlying features between the data, leading to a model that will focus on learning from the noise in the data. And between models using 64 or 32 neurons per hidden layer, the results are the same, and using one or the other is totally justified.

Table 5.5: Comparison of Models' performances on test data with same hyperparameters except the number of neurons per hidden layer. Performance are computed on test data. The results here are more complex. Although the one with 128 neurons per hidden layer performs slightly better in MAE and MSE, the R^2 is lower. This may be due to a lack of data. The model is more complex, but the amount of data doesn't allow the model to capture the underlying features between the data, leading to a model that will focus on learning from the noise in the data. And between models using 64 or 32 neurons per hidden layer, the results are the same, and using one or the other is totally justified.

Error Metrics	Model with 32 neurons	Model with 64 neurons	Model with 128 neurons
Test MSE	2.83	2.81	2.51
Test MAE	0.89	0.85	0.81
Test R^2	0.53	0.48	0.18

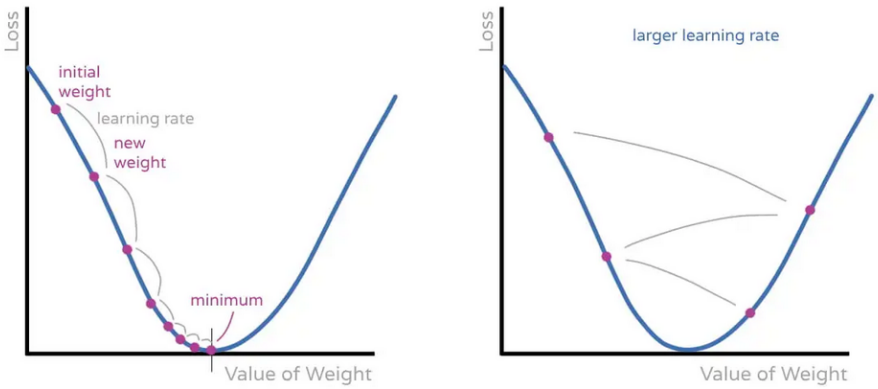


Figure 5.6: Effect of a big learning rate on the training. The learning rate is the "step" of actualization of the weights of the NN. If it is too big, the "step" can be too big, missing the minimum of loss and give weights values that could create a loss far bigger than the one seen before.

Another comparison can be made with regard to the number of epochs the model is trained for. The results are shown in table 5.6. At first glance, the MAE and MSE results of the two models trained for 100 and 200 epochs are better than the results for the model trained for 150 epochs. But the difference lies in the value of R^2. In the case of the model trained for 100 epochs, the negative R^2 indicates that the model has not captured any underlying traits in the data, which in this situation is a sign of underfitting, as the model has not been trained for enough epochs. For the model trained for 200 epochs, the R^2 is lower than that trained for 150 epochs, which can be interpreted as one of the first signs of overfitting. The model begins to learn more about the noise in the data than the underlying features. After

Table 5.6: Comparison of Models' performances on test data with same hyperparameters except the number of epochs. Although the MAE and MSE results of the two models trained over 100 and 200 epochs are better than the results of the model trained over 150 epochs, the difference will come down to the R^2 value. In the case of the model trained for 100 epochs, the negative R^2 indicates that the model has not captured any underlying traits in the data, which in this situation is a sign of underfitting, as the model has not been trained for enough epochs. For the model trained for 200 epochs, the R^2 is lower than that trained for 150 epochs, which can be interpreted as one of the first signs of overfitting. The model begins to learn more about the noise in the data than the underlying features.

Error Metrics	Model with 100 epochs	Model with 150 epochs	Model with 200 epochs
Test MSE	2.71	2.81	2.63
Test MAE	0.85	0.85	0.83
Test R^2	-0.11	0.48	0.33

optimizing the model for the knee flexion, the hyperparameters use for that model were also used for the models of the 2 other DoFs.This gives the results shown in table 5.7, and the functions of the loss curves can be seen in fig. 5.8. The loss curves show no anomalies. And the results in table 5.7 are quite similar for the same combination of hyperparameter, as the results for Knee flexion and hip flexion are quite the same. The results for hip abduction can be explained by the fact that the angle varies in [-8°, 2°], a 10° amplitude variation as the hip flexion varies in [-10°, 25°] for a 35° amplitude and knee flexion in [10°, 70°] for a 60° amplitude. Adding that with the formula for MAE eq. (5.2) and MSE eq. (5.1), the error on angle will be generally less for the hip abduction given his amplitude of variation. Showing that the hypothesis that the model's hyperparameters for one DoF fit for the others DoFs.

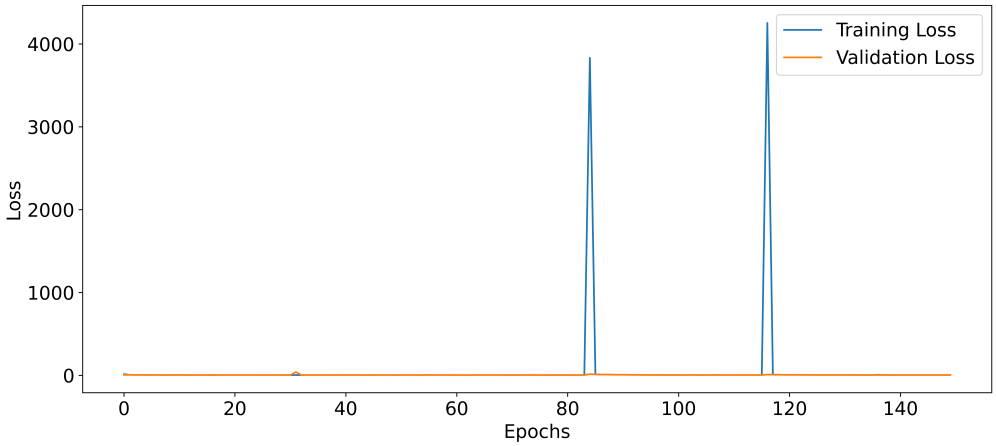


Figure 5.7: Representation of a bad loss model. The hyperparameters are the same than the model with normalized gait phase in table 5.3 but the learning rate is 0.01. Here the profile of loss are not the one that should be obtain for a good model. There are no real convergence and there are peaks of training loss. This phenomenon is explained by fig. 5.6. As the learning rate is bigger, the actualization of the weigths can miss the value for a minimum loss and even go to values where the loss is far bigger.

Table 5.7: Performance metrics on test data for the models of different DoF in the case of the restricted model and a smaller population

DoFs	Test MSE	Test MAE	Test R^2	Training time
Knee flexion	2.81	0.85	0.48	416.55 s
Hip flexion	2.62	0.61	0.45	368.85 s
Hip abduction	0.34	0.3	0.64	394.26 s

However, it should be pointed out that at the output of the neural network, there are two kinds of quantities. For each key point, there is the normalized gait phase and the corresponding angular position in degrees. The error on these two quantities is contained in the MSE and MAE values in table 5.7, without being able to distinguish between them. In order to better analyze this error, the prediction is now individually evaluated for the angle and for the gait phase.

In table 5.8, the mean and standard deviation of the RMSE on the normalized gait phase in range [0, 1] can be observed for the 3 DoFs. The mean RMSE in range [0.01, 0.022] can be interpreted as in average, considering a gait period of one second, the temporal position of a predicted key point will be in a range of [0.01 s, 0.022 s] around the real position of the key point. fig. 5.9, the mean and standard deviation of the error made on the normalized gait percentage during the prediction of key points classified by walking speed is shown. This plot is done to observe what could be the worse values of this RMSE on the normalized gait phase for the 3 DoFs. And there are some cases where this RMSE could be of 0.045, a the predicted key point, considering a gait period of 1 second, could have an offset of 0.045 s, 45 ms on the real key point.

In table 5.9, can be observed the mean and standard deviation of the relative RMSE on the angular position of the key points. The relative RMSE is the RMSE between the real and predicted key points divided by the amplitude of the curve. With that it is easier to represent if the error made is consequent or not. In this case, the average RMSE made on the key points represent [3.5%, 5,8%] of their respective curve amplitude. In fig. 5.10, the mean and standard deviation of the error made on the angular position during the prediction of key

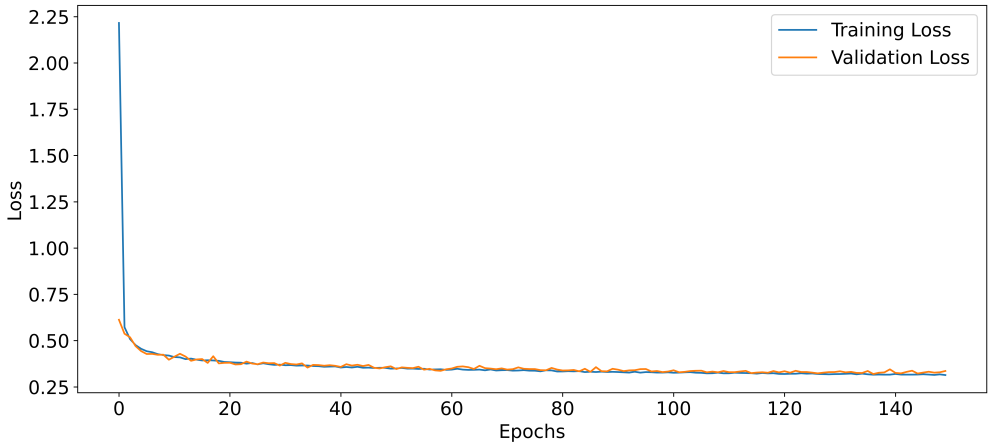
Table 5.8: Mean and Standard deviation of the RMSE on normalized gait phase on all data for the reduced models of different DoFs

DoFs	Mean RMSE	Standard deiviation RMSE
Knee flexion	0.017	0.008
Hip flexion	0.022	0.013
Hip abduction	0.010	0.005

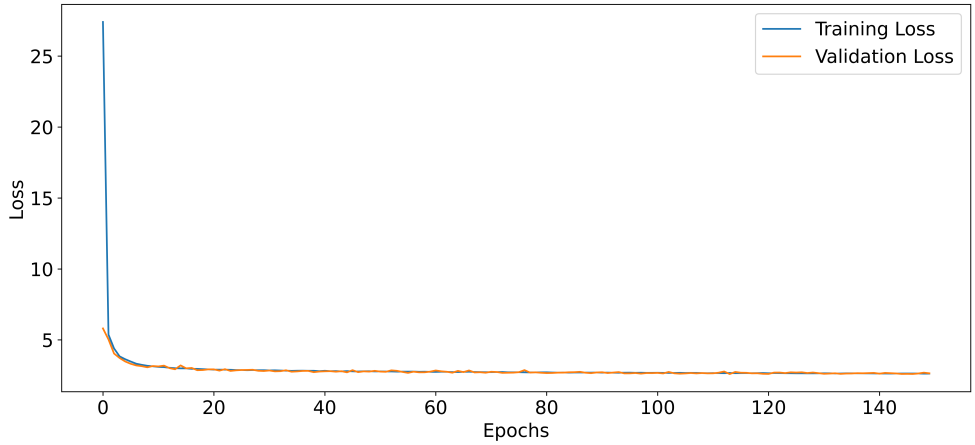
points as a function of walking speed is shown. The choice of representing as a funtion of the walking speed is more arbitrary, as the objective is more to see what could be in general the worst case scenario. For the 3 DoFs, The worst case scenario in the majority of case is an error 15% of the amplitude.

Table 5.9: Mean and Standard deviation of th Relative RMSE on angular position of the DoF on all data for the reduced models of different DoFs Relative RMSE is the RMSE between the angular position between the predicted and the real key points divided by the amplitude of the real curve.

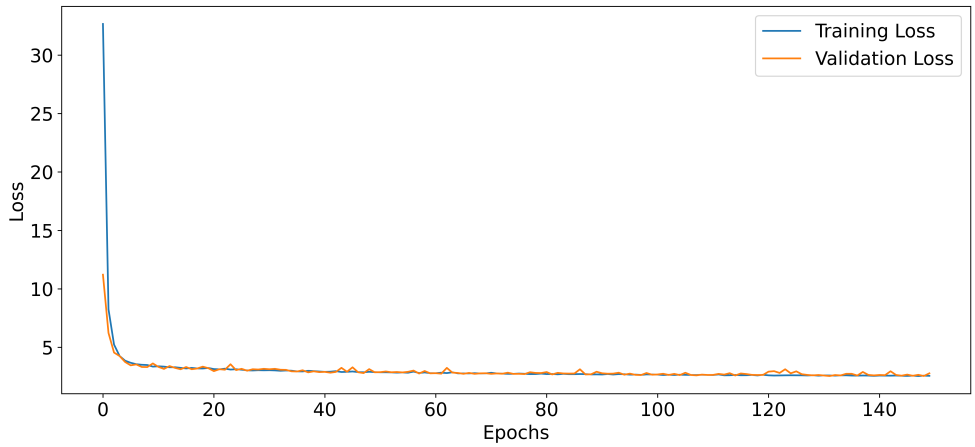
DoFs	Mean RMSE	Standard deiviation RMSE
Knee flexion	0.035	0.019
Hip flexion	0.043	0.032
Hip abduction	0.058	0.038



(a) Loss for the hip abduction in a smaller population

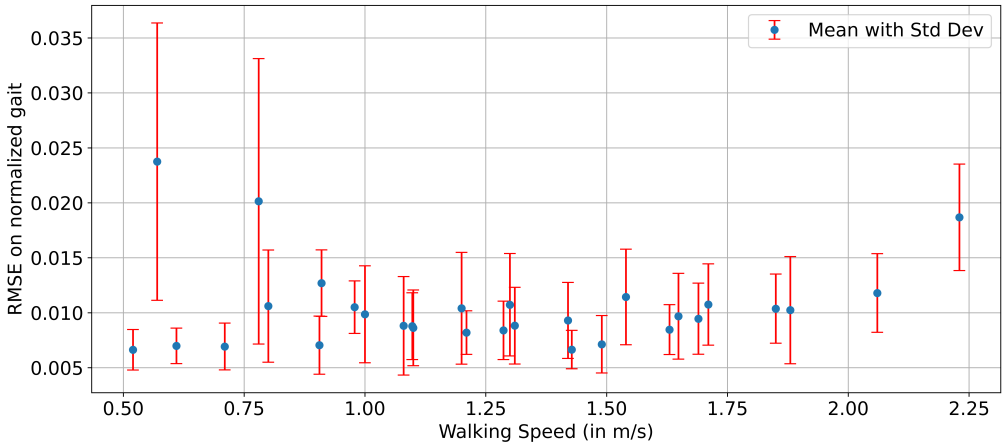


(b) Loss for the hip flexion in a smaller population

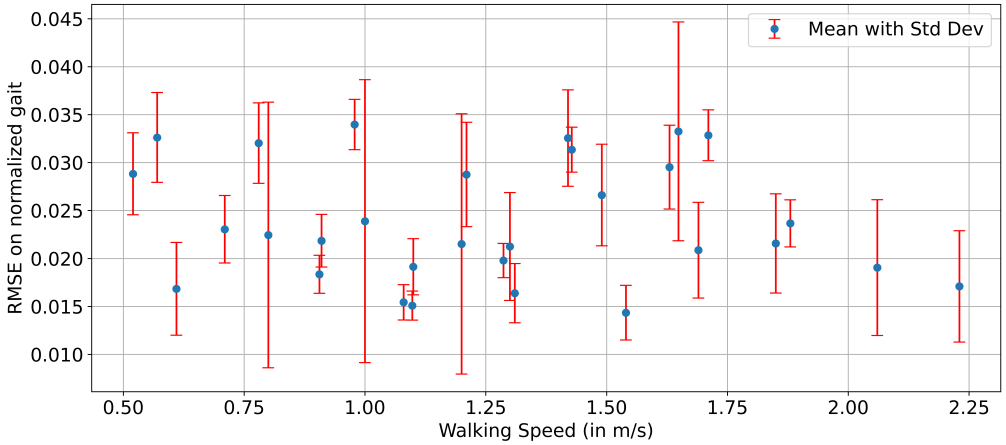


(c) Loss for the knee flexion in a smaller population

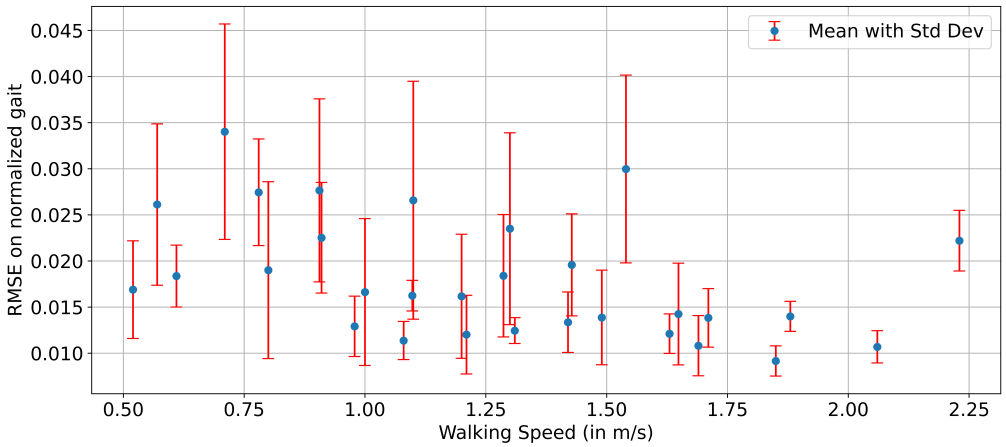
Figure 5.8: Representation of the loss curve for the reduced model fig. 5.5 of the 3 DoFs. For a limited population (age < 25, male), we obtain these Loss results based on the MSE during the training phase. For fig. 5.8a, we have a final loss in training of 0.3. fig. 5.8b, the final loss in training is about 2.7. fig. 5.8c, the final loss in traing is about 2.6.



(a) Error on normalized gait percentage of the predicted key points for hip abduction

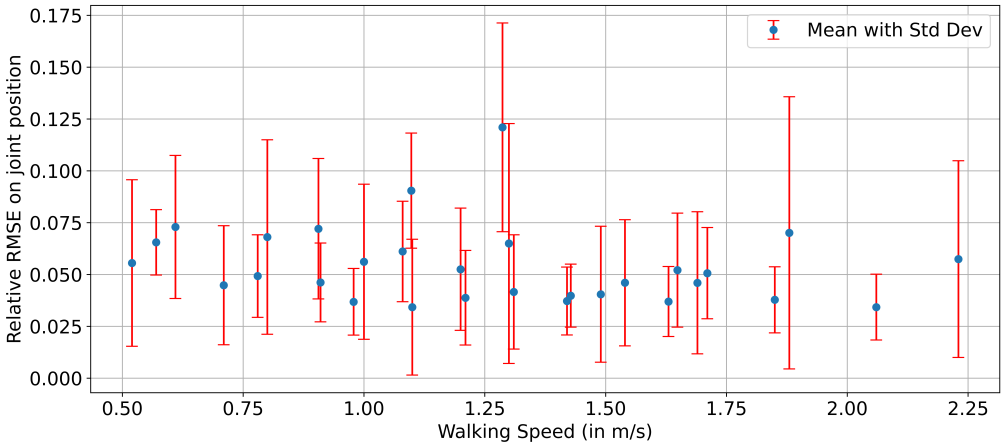


(b) Error on normalized gait percentage of the predicted key points for hip flexion

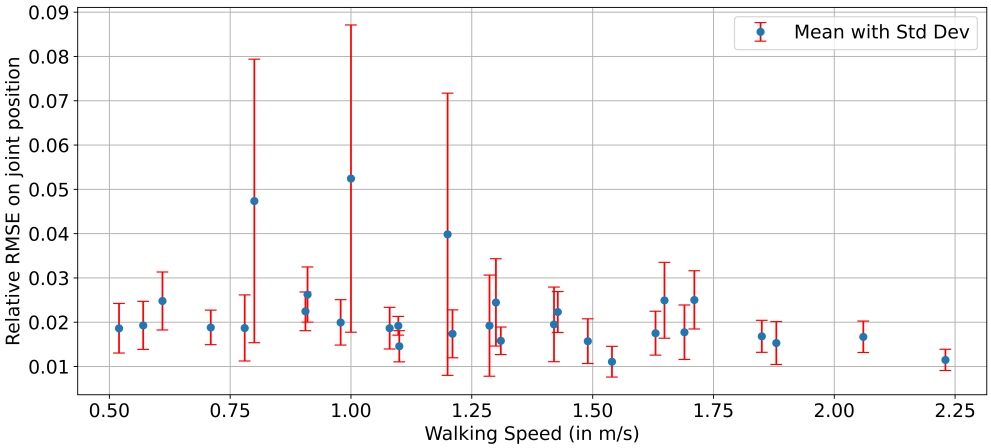


(c) Error on normalized gait percentage of the predicted key points for knee flexion

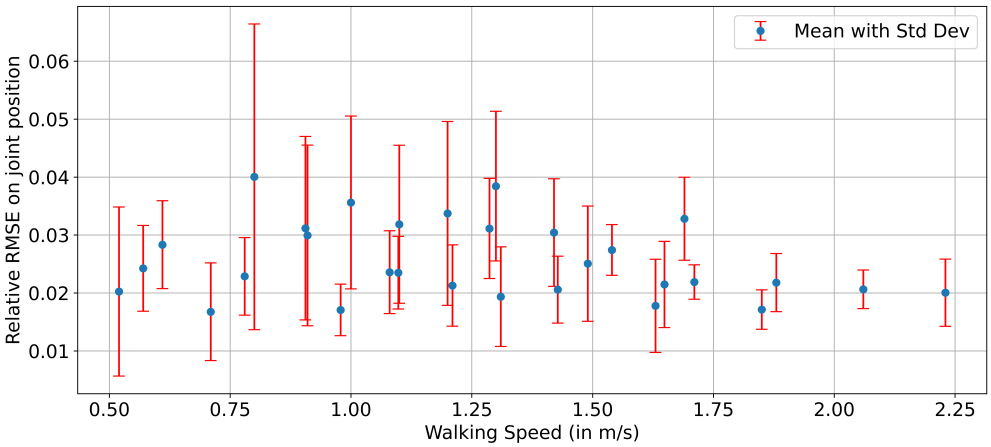
Figure 5.9: Representation of the error made on the normalized gait phase in [0, 1] when predicting the key points with the reduced model fig. 5.5 as function of the walking speed. Each of the figue represent for a DoF, give a specific walking speed, the mean and standard deviation of the error made on the normalized gait phase in [0, 1] when predicting the key points. So considering a gait period of 1 sec, if the error is 0.01, the predicted key point will be in a 0.01 second range around the real position of the key point.



(a) Error on angular position of the predicted key points for hip abduction



(b) Error on angular position of the predicted key points for hip flexion



(c) Error on normalized gait percentage of the key points for knee flexion

Figure 5.10: Representation of the relative RMSE made on the angular position when predicting the key points with the reduced model fig. 5.5 as function of the walking speed. Each of the figue represent for a DoF, give a specific walking speed, the mean and standard deviation of the error made on the angular position when predicting the key points. This relative error is the RMSE betwee the predicted and the real key points divided by the amplitude (max - min) of the real curve.

5.4.3 Complete Neural Network

Having verified that the idea of such a model for estimating key points is functional in the first step, the next step is to train the actual model shown in fig. 4.3 for each of the DoFs. In this case too, training will be carried out with the normalized version of the gait phase as output. The number of samples here is 308520 (5142*60). At the end of training for the knee flexion model, we obtain the following hyperparameters for the model:

- Number of hidden layers: 7
- batch size: 32
- epochs: 150
- Neurons per hidden layers: 64
- Dropout rate: 0.0
- Activation function: LeakyReLU
- Regularization function: None
- Initialization: He normal

Here again, the hyperparameters obtained for training of the knee flexion model were reused for the rest of the DoFs models. By doing that, it can be observed in fig. 5.11 that the 3 loss converges. The values for hip flexion and knee flexion loss with the test data are similar. The difference with hip abduction is the range of values which is bigger for the knee flexion and hip flexion than for the hip abduction. This is confirmed by fig. 5.11, where the loss curves for the 3 DoF models clearly show that an optimum has been reached in all 3 cases. Test results for this model are shown in table 5.10. As in the previous step, we need to better understand

Table 5.10: Performance metrics for the models of the different DoFs for the complete model and the entire population

DoFs	Test MSE	Test MAE	Test R^2	Training time (in s)
Knee flexion	3.47	0.93	0.47	2719.20
Hip flexion	3.57	0.74	0.53	2719.20
Hip abduction	0.7	0.38	0.39	2440.78

the meaning of the error calculated by Loss and MAE, since it comes from two contributions: the estimation of the normalized gait phase of the key points, and the estimation of their angular position. table 5.11 presents the mean and standard deviation of the RMSE of the normalized gait phase for this complete model. In average, for a gait period of one second, there will be an offset of 0.024s between the predicted key points and the real key points. table 5.12 give the results on the relative RMSE on the angular position error of the key points. The relative RMSE is the RMSE between the predicted and real key points divided by the amplitude of the real curve. And in average, the difference in angular position between the real key points and the predicted key points is less than 8% of the amplitude of the curve.

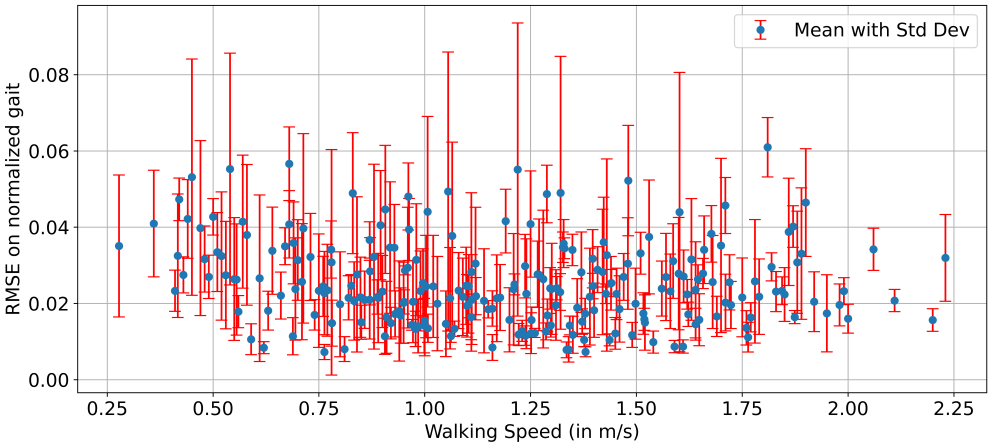
To observe the worst cases errors in the majority, the fig. 5.12 and fig. 5.13 were made using the entire database to make these estimates. In particular, it can be seen that an average error of between 0.02 and 0.03 is made on the normalized gait phase when predicting key points. Also, the average error on the angular position of key points is lower for hip abduction (0.55 on average) than for the other two DoFs, as might be expected from the MAE and Loss values.

Table 5.11: Mean and Standard deviation of the RMSE on normalized gait phase on all data for the complete model of different DoFs

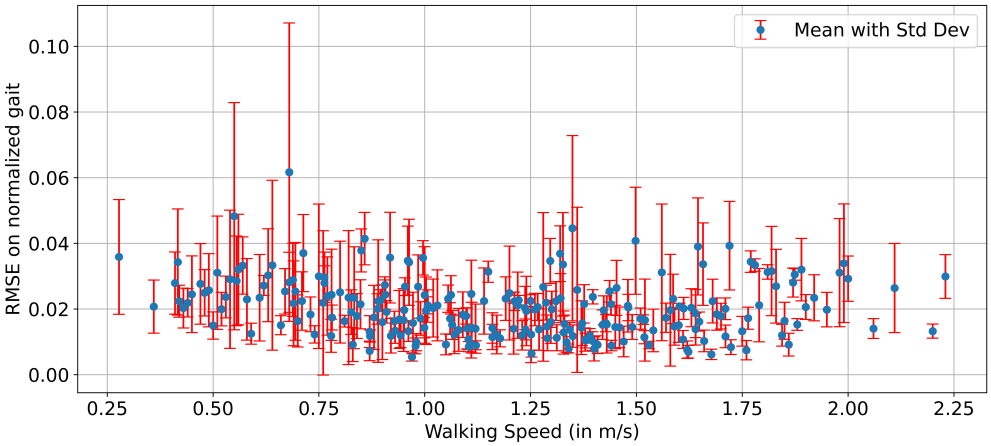
DoFs	Mean RMSE	Standard deiviation RMSE
Knee flexion	0.028	0.014
Hip flexion	0.024	0.015
Hip abduction	0.021	0.014

Table 5.12: Mean and Standard deviation of th Relative RMSE on angular position of the DoF on all data for the complete model of different DoFs Relative RMSE is the RMSE between the angular position between the predicted and the real key points divided by the amplitude of the real curve.

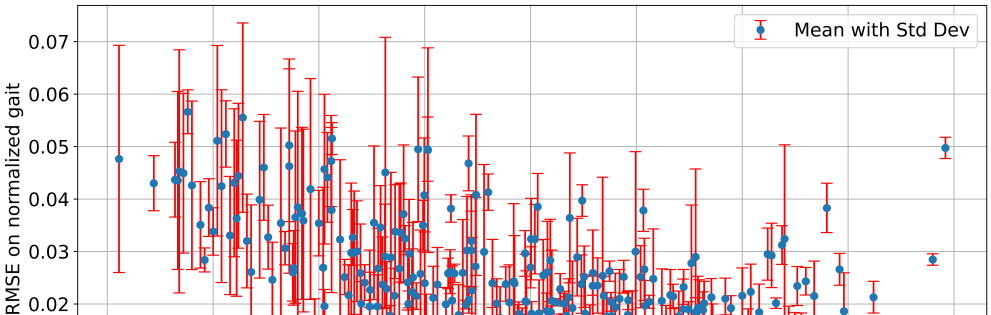
DoFs	Mean RMSE	Standard deiviation RMSE
Knee flexion	0.037	0.021
Hip flexion	0.051	0.037
Hip abduction	0.079	0.056

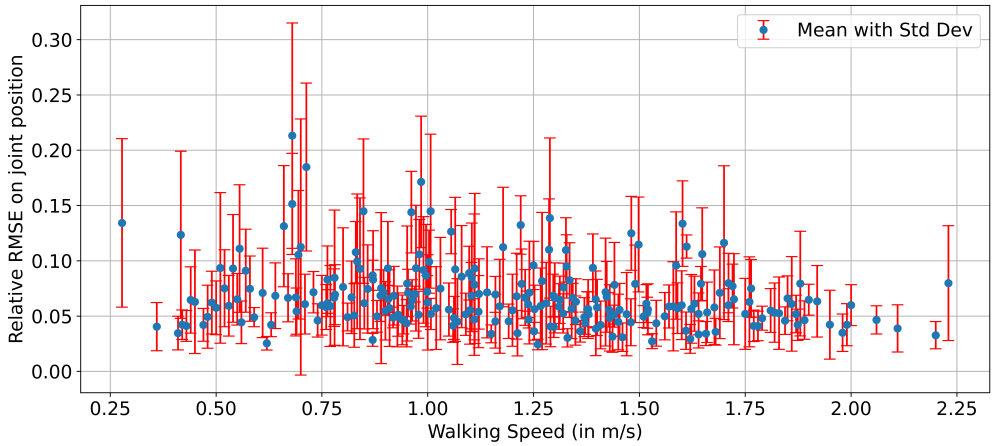


(a) Error on normalized gait percentage of the predicted key points for hip abduction

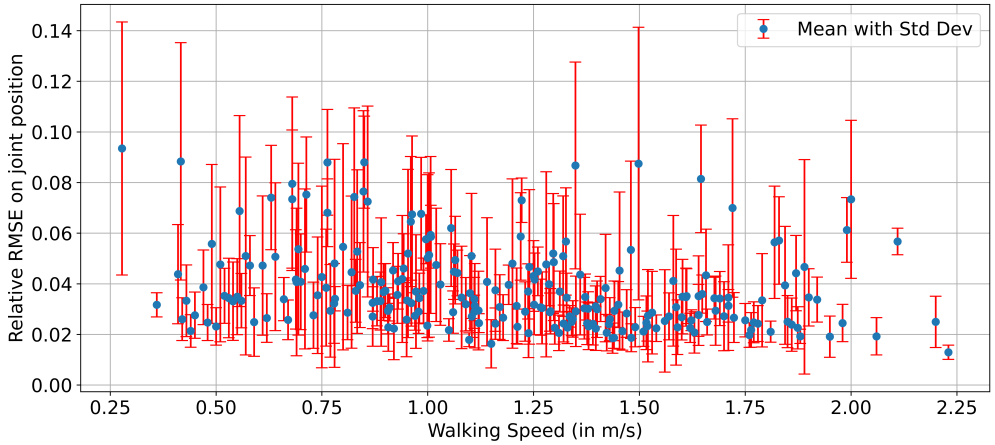


(b) Error on normalized gait percentage of the predicted key points for hip flexion

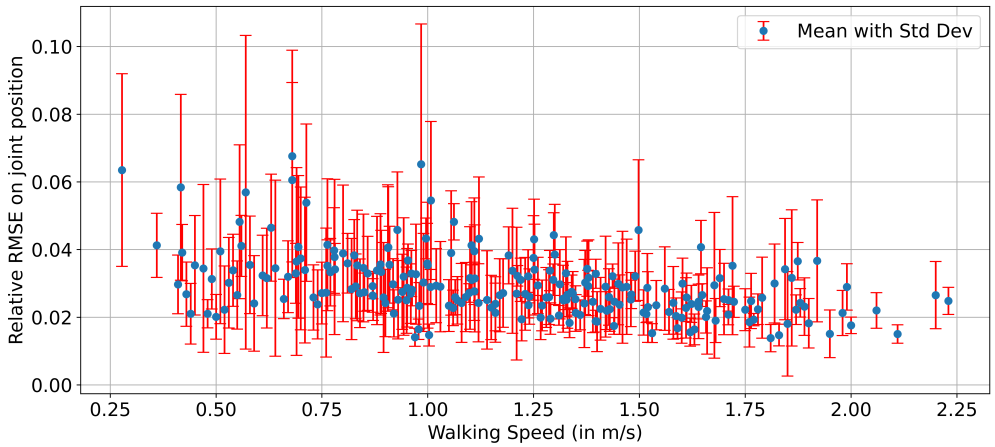




(a) Error on angular position of the predicted key points for hip abduction



(b) Error on angular position of the predicted key points for hip flexion



(c) Error on normalized gait percentage of the key points for knee flexion

Figure 5.13: Representation of the error made on the angular position when predicting the key points with the complete model fig. 4.3 in fonction of the walking speed. Each of the figue represent for a DoF, give a specific walking speed, the mean and standard deviation of the error made on the angular position when predicting the key points.

5.5 Trajectory Generation Performance and Optimization

After assessing above during training, two things were noticed:

- For training, the key points are classified according to their normalized gait phase in ascending order. However, after various tests, the relative position of these key points in this ranking is not fixed. In particular, between two extremities, it can happen that the present inflection point and an added point can see their relative position in the ranking exchanged, because the normalized gait phase of the added point is relative to the normalized gait phases of the extremities, and not to that of the inflection point. As a result, the gait phase of the inflection point may be higher or lower than that of the added point. This results in output neurons learning data from different key points, which is not good for training.

- In the same case where using the normalized gait phase [0,1] instead of the gait percentage [0,100] reduced the size of the neural network, it's possible that changing the angles in radian from a scale of [-25°, 70°] to [-3.14 rad, 3.14 rad] would reduce the size of the neural network even further.

But before applying these modifications, we need to look at the results of trajectory generation, i.e. the combination of key point generation and interpolation.

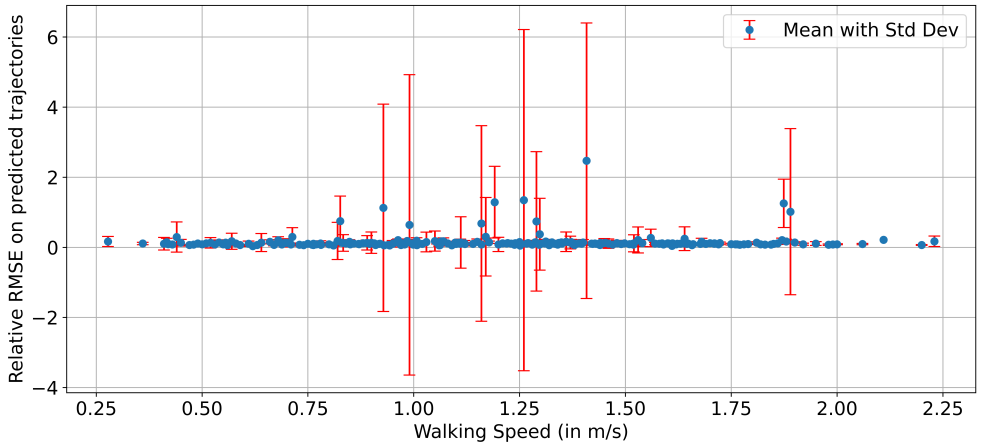
5.5.1 Trajectory Generation Initial Complete Model Performance

Using data from the entire database, the results of the relative RMSE of the predicted trajectories have been calculated and presented in table 5.13. Although the average of these RMSEs gives satisfactory values, i.e. a maximum of 11% over the 3 DoFs of the amplitude of the real curve, the standard deviation values are quite high compared with the average and are predictive of fairly substantial outliers. This can be seen from figure fig. 5.14. Although a large proportion of the values are close to the mean, there are some rather aberrant cases, with relative RMSEs as high as 6000% of amplitude.

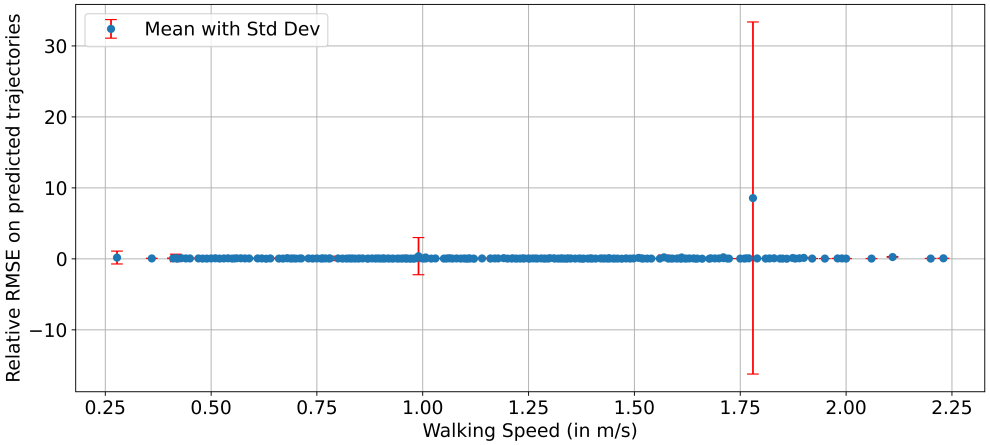
The representation of the worst-case scenario prediction result can be observed in fig. 5.15. The curves are absurd and even pose a danger to the user and the exoskeleton. The best-case scenario can also be observed in fig. 5.16, with relative RMSEs below 1% in each case. Additionally, the proportion of predicted trajectories with a relative RMSE of more than 10% was evaluated. It is 21.5% for the knee flexion model, 12.6% for the hip flexion model, and 39.2% for the hip abduction model. These results are significant considering that they include those with absurd values as seen previously.

Table 5.13: Mean and Standard deviation of th Relative RMSE on the predicted trajectories of the DoF on all data for the complete model of different DoFs. After doing the Spline interpolation between the predicted key points, it is possible to evaluate the model at the same normalized gait phase as the points from the real curve. Then the RMSE between those evaluated points and the points from the real curve can be compute. And to obtain the relative RMSE it is divided by the real curve amplitude.

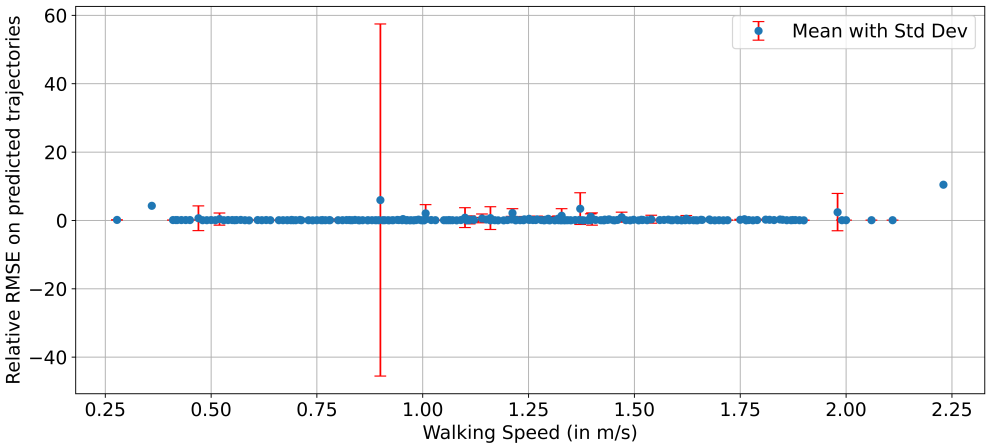
DoFs	Mean RMSE	Standard deiviation RMSE
Knee flexion	0.11	2.09
Hip flexion	0.074	0.57
Hip abduction	0.10	0.32



(a) Error on predicted trajectories for hip abduction

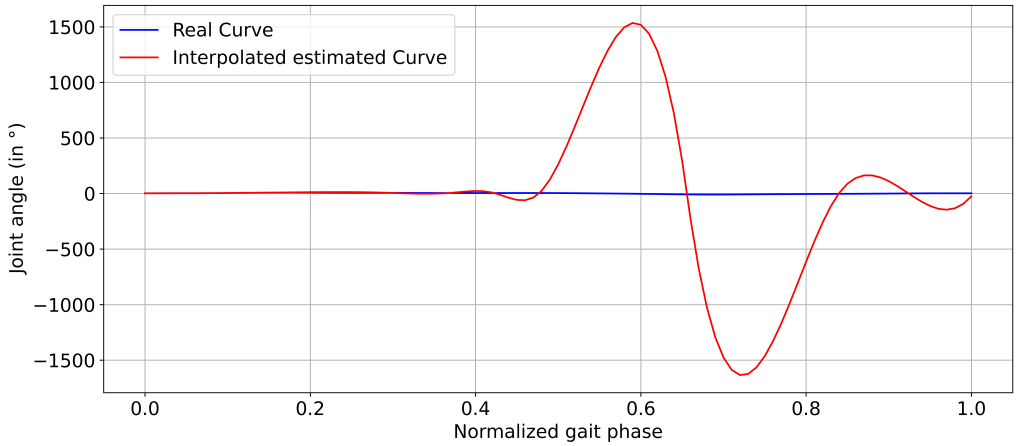


(b) Error on predicted trajectories for hip flexion

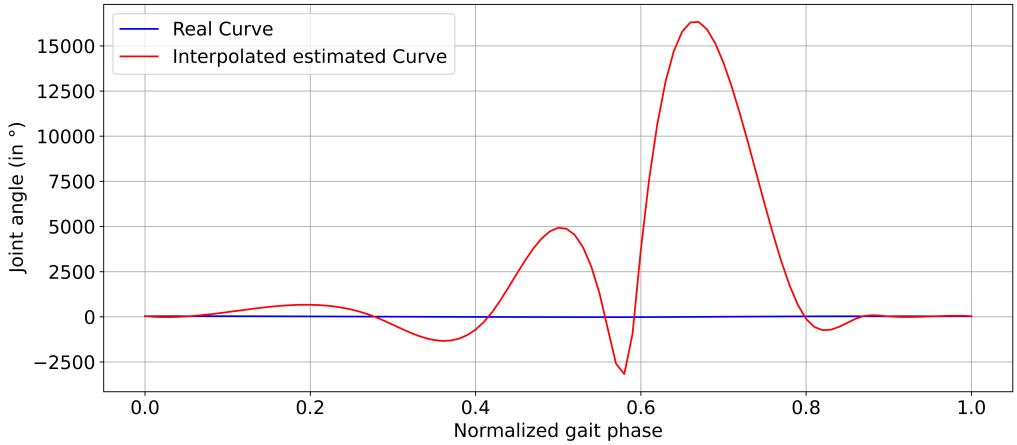


(c) Error predicted trajectories for knee flexion

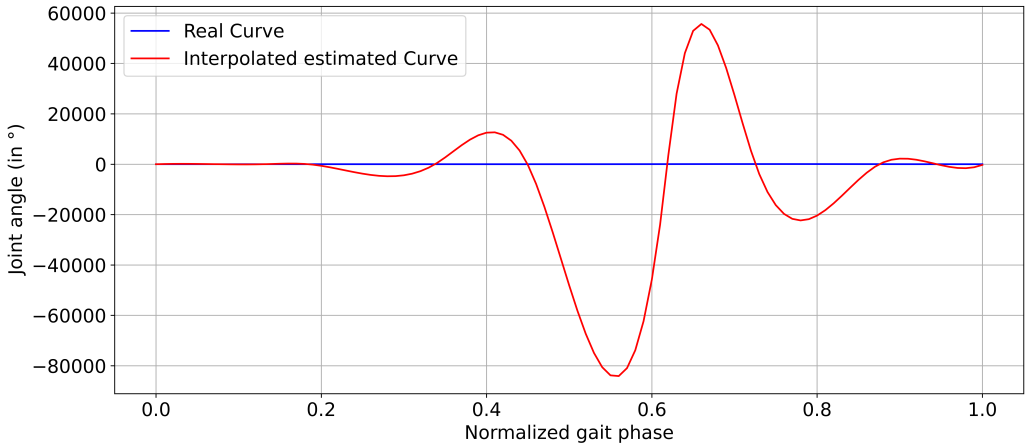
Figure 5.14: Representation of the relative RMSE of the predicted trajectories with the complete model fig. 4.3 as function of the walking speed. Each of the figue represent for a DoF, give a specific walking speed, the mean and standard deviation of the error made on the angular position when predicting the key points. The objective is to observe the worse cases scenario for this errors.



(a) Worst predicted trajectory for Hip abduction

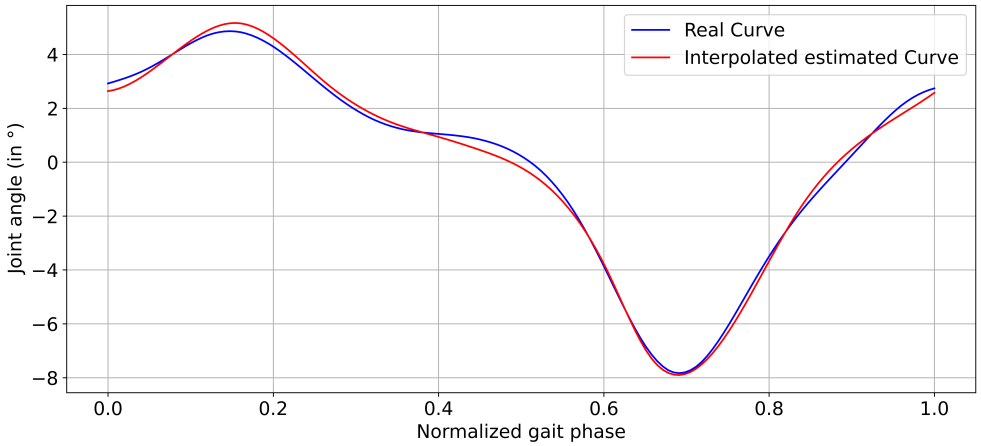


(b) Worst predicted trajectory for Hip flexion

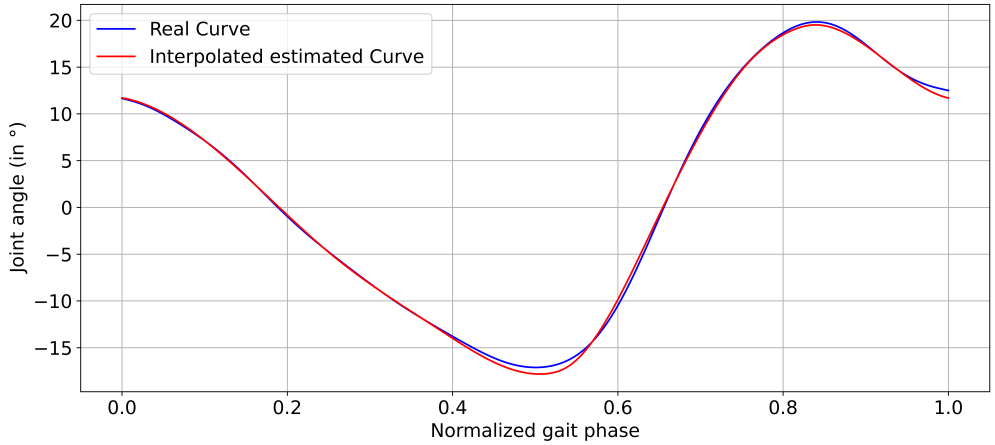


(c) Worst predicted trajectory for Knee flexion

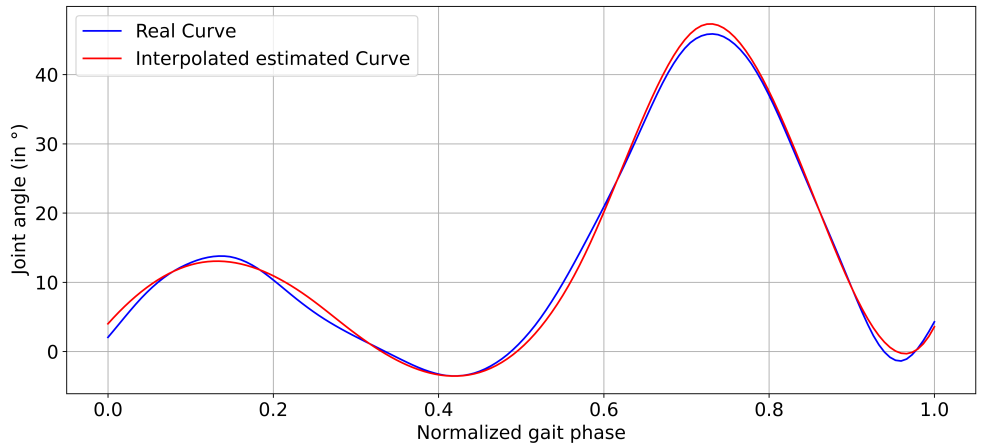
Figure 5.15: Representation of the worst predicted trajectories with the complete model for the 3 DoF fig. 4.3. The relative RMSE for this worst prediction are: 452.19 for the knee flexion, 99.59 for the hip flexion and 35.72 for the hip abduction.



(a) Best predicted trajectory for Hip abduction



(b) Best predicted trajectory for Hip flexion



(c) Best predicted trajectory for Knee flexion

Figure 5.16: Representation of the best predicted trajectories with the complete model for the 3 DoF fig. 4.3. The relative RMSE for this best prediction are: 0.018 for the knee flexion, 0.008 for the hip flexion and 0.018 for the hip abduction.

5.5.2 Model Optimization and Trajectory Generation Performance

For this model optimization, changes to the key points will be applied while respecting the original intention of some of the points, which was to improve interpolation in a specific area. The new key point values will be discussed in the following paragraphs.

Knee Flexion New Key points

The new list of key points keeps many similarities with the original list, just the added points are changed. The rule for the choice of inflection points stay the same. The list is:

- A maximum at e_{KF1} within the interval [0%, 20%] of gait cycle
- A minimum at e_{KF2} within the interval [25%, 60%] of gait cycle
- A maximum at e_{KF3} within the interval [65%, 90%] of gait cycle
- e_{KFA} at gait percentage 0
- e_{KFZ} at gait percentage 100
- u_{KF1} the inflection point between e_{KF1} and e_{KF2}
- u_{KF2} the inflection point between e_{KF2} and e_{KF3}
- The point at $x_{KF1} = \frac{e_{KFA} + e_{KF1}}{2}$. It serves to better interpolate the first part of the curve in the interval $[e_{KFA}, e_{KF1}]$.
- The two points at $x_{KFi+1} = \frac{e_{KFi}+u_{KFi}}{2}$ with i = 1,2. They have been modify to ensure that the gait percentage of x_{KFi+1} is always inferior to the gait percentage of u_{KFi} for i = 1,2. They serve to add another point with the respective inflection points in the intervals $[e_{KFi}, u_{KFi}]$ for a better interpolation.
- Lastly, 2 final points in the interval $[e_{KF3}, e_{KFZ}]$. The first is fixed at gait percentage $x_{KF4} = \frac{e_{KF3}+e_{KFZ}}{2}$. The second one, at gait percentage $x_{KF5} = 0.2.e_{KF3}+0.8.e_{KFZ}$, obtained by trial and error, as it has generally been noted that there is an approximation error around this chosen gait phase.

This choice is made to maintain this order for the key points in terms of gait percentage: $[e_{KFA}, x_{KF1}, e_{KF1}, x_{KF2}, u_{KF1}, e_{KF2}, x_{KF3}, u_{KF2}, e_{KF3}, x_{KF4}, x_{KF5}, e_{KFZ}]$.

Hip Flexion New Key points

The new list as in the precedent case just came with certain modifications to some added points to maintain a certain order. The new list is:

- A minimum at e_{HF1} within the interval [40%, 65%] of gait cycle
- A maximum at e_{HF2} within the interval [70%, 90%] of gait cycle
- e_{HFA} at gait percentage 0 and e_{HFZ} at gait percentage 100
- u_{HF1} the inflection point between e_{HF1} and e_{HF2}
- Three points in the interval $[e_{HFA}, e_{HF1}]$ which are in order: $\frac{x_{HF1}=9.e_{HFA}+e_{HF1}}{10}$, $x_{HF2} = \frac{e_{HFA}+e_{HF1}}{2}$, $x_{HF3} = \frac{e_{HFA}+3.e_{HF1}}{4}$.

- One points in the interval $[e_{HF1}, u_{HF1}]$ which are in order: $x_{HF4} = \frac{9.e_{HF1}+e_{HF2}}{10}$
- Two points in the interval $[u_{HF1}, e_{HF2}]$ which are $x_{HF5} = \frac{u_{HF1}+3.e_{HF2}}{4}$, $x_{HF6} = \frac{u_{HF1}+9.e_{HF2}}{10}$.
- One point in the interval $[e_{HF2}, e_{HFZ}]$ which is: $x_{HF7} = \frac{e_{HF1}+e_{HF2}}{2}$

This choice is made to maintain this order for the key points in terms of gait percentage: $[e_{HFA}, x_{HF1}, x_{HF2}, x_{HF3}, e_{HF1}, x_{HF4}, u_{HF1}, x_{HF5}, x_{HF6}, e_{HF2}, x_{HF7}, e_{HFZ}]$.

Hip Abduction New Key points

The new list as in the precedent case just came with certain modifications to some added points to maintain a certain order. The new list is:

- A maximum at e_{HA1} within the interval [5%, 30%] of gait cycle
- A minimum at e_{HA2} within the interval [55%, 80%] of gait cycle
- e_{HAA} at gait percentage 0 and e_{HAZ} at gait percentage 100
- u_{HA1} the inflection point between e_{HA1} and e_{HA2}
- One point in the interval $[e_{HAA}, e_{HF1}]$ which is: $x_{HA1} = \frac{e_{HAA}+e_{HF1}}{2}$.
- The 4 points in the interval $[e_{HF1}, u_{HA1}]$ which are in order: $x_{HA2} = \frac{3.e_{HF1}+u_{HA1}}{4}$, $x_{HA3} = \frac{3.e_{HF1}+2.u_{HA1}}{5}$, $x_{HA4} = \frac{e_{HF1}+u_{HA1}}{2}$, $x_{HA5} = \frac{2.e_{HF1}+3.u_{HA1}}{5}$.
- Two points in the interval $[e_{HF2}, e_{HAZ}]$ which are in order: $x_{HA6} = \frac{e_{HF2}+end\_point}{2}$, $\frac{x_{HA7}=e_{HF2}+4.e_{HAZ}}{5}$.

This choice is made to maintain this order for the key points in terms of gait percentage: $[e_{HAA}, x_{HA1}, e_{HA1}, x_{HA2}, x_{HA3}, x_{HA4}, x_{HA5}, u_{HA1}, e_{HA2}, x_{HA6}, x_{HA7}, e_{HAZ}]$.

Optimized Model Configuration and Results

The model was train with the new key points and the new configuration, which is with the normalized gait phase and the angle in radian. The training was done with 257100 samples (5142*50). At the end of the GridSearch for the knee flexion model, the optimized combination of hyperparameters is:

- Number of hidden layers: 3
- batch size: 32
- epochs: 150
- Neurons per hidden layers: 64
- Dropout rate: 0.0
- Activation function: LeakyReLU
- Regularization function: None
- Initialization: He normal

The modifications have halved the number of hidden layers required for the system, significantly reducing the size of the neural network, which will have observable impacts as noted below, while still achieving similar convergence as seen in fig. 5.17.

table 5.14 presents the results of this new version of the model, noting that the hyperparameters determined for the knee flexion model have been reused for the other DoFs. Although the MSE and MAE are difficult to compare since the range of output values is no longer the same, generally shifting from [-25°, 70°] to [-3.14 rad, 3.14 rad], a comparable value is R^2. In the new configuration, the model captures the variability in the data better. Also the training time is also diminish, but a direct comparison can't be done because the number of samples used is also different. In this case as well, since the error calculated with MAE and MSE includes both the error in the normalized gait phase and the error in the angular position, it is also necessary to calculate these two errors individually.

The results obtained individually for the normalized gait phase and angular position are presented in table 5.16 and table 5.15. On average, the results are quite similar for the generation of key points. This changes slightly with the results for the predicted trajectories in Table table 5.17. Here, the results have slightly improved on average, but the most significant change is in the maximum relative RMSE, which is generally lower than for the previous complete model. This is also evident in fig. 5.18, which, compared to fig. 5.14, shows that the outliers no longer impact the average values as much, although they still exist. The percentage of trajectories with a relative RMSE less than 10% has been significantly reduced to less than 7%.

Examples of the best and worse predicted trajectories can be seen in fig. 5.19 and fig. 5.20.

Table 5.14: Performance metrics on the test data for the modified complete models of the different DoFs for the complete model

DoFs	Test MSE	Test MAE	Test R^2	Training time (in s)
Knee flexion	0.0014	0.0242	0.65	1941.46
Hip flexion	0.0015	0.0208	0.66	1944.28
Hip abduction	0.0004	0.0128	0.64	1923.28

Table 5.15: Mean, Standard deviation and Maximum of th Relative RMSE on angular position of the DoF on all data for the modified complete model of different DoFs Relative RMSE is the RMSE between the angular position between the predicted and the real key points divided by the amplitude of the real curve.

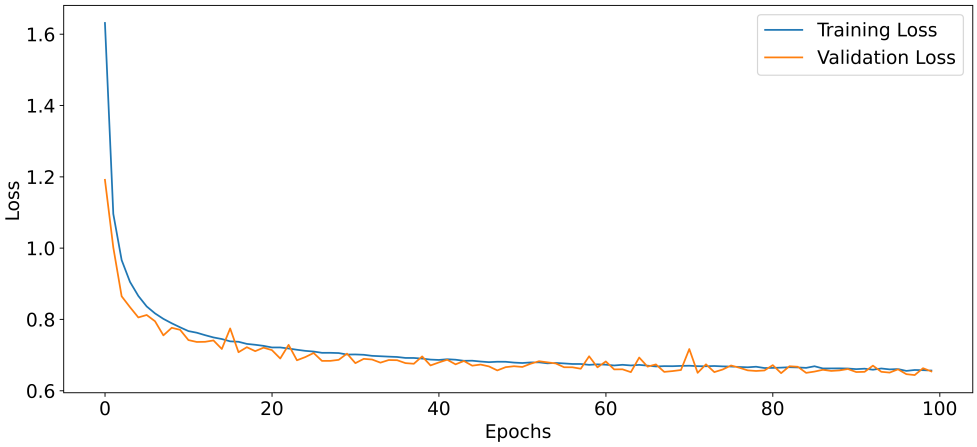
DoFs	Mean RMSE	Standard deviation RMSE	Max RMSE
Knee flexion	0.038	0.021	0.28
Hip flexion	0.055	0.038	0.41
Hip abduction	0.088	0.061	0.88

Table 5.16: Mean, Standard deviation and Maximum of the RMSE on normalized gait phase on all data for the modified complete model of different DoFs

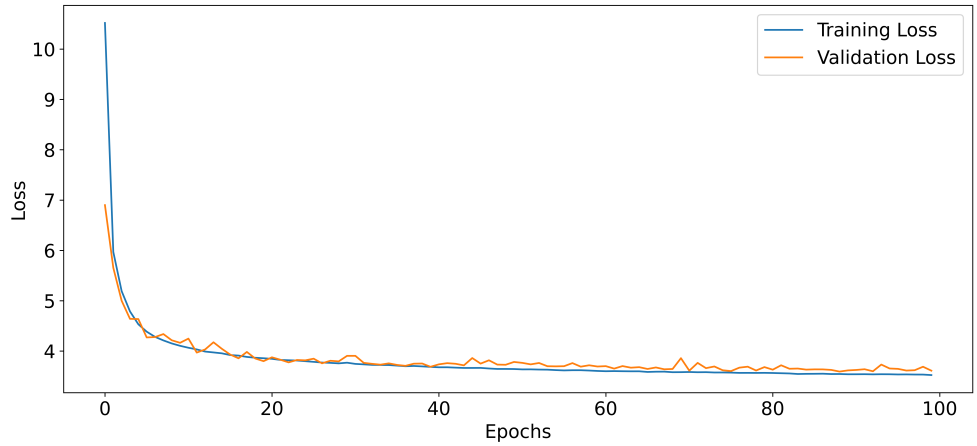
DoFs	Mean RMSE	Standard deviation RMSE	Max RMSE
Knee flexion	0.022	0.014	0.14
Hip flexion	0.0205	0.017	0.11
Hip abduction	0.014	0.014	0.23

Table 5.17: Mean and Standard deviation of th Relative RMSE on the predicted trajectories of the DoF on all data for the modified complete model of different DoFs. After doing the Spline interpolation between the predicted key points, it is possible to evaluate the model at the same normalized gait phase as the points from the real curve. Then the RMSE between those evaluated points and the points from the real curve can be compute. And to obtain the relative RMSE it is divided by the real curve amplitude.

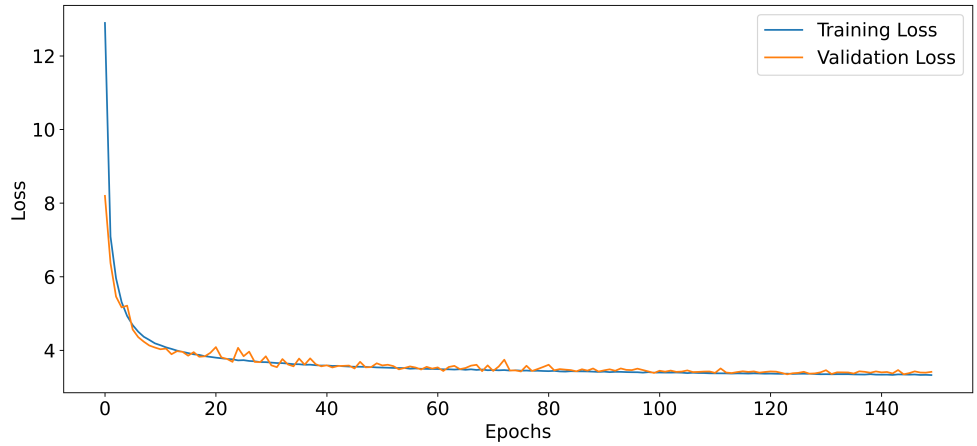
DoFs	Mean RMSE	Standard deiviation RMSE	Mas RMSE
Knee flexion	0.053	0.028	0.21
Hip flexion	0.058	0.027	0.72
Hip abduction	0.098	0.058	1.04



(a) Loss for the complete model of the hip abduction

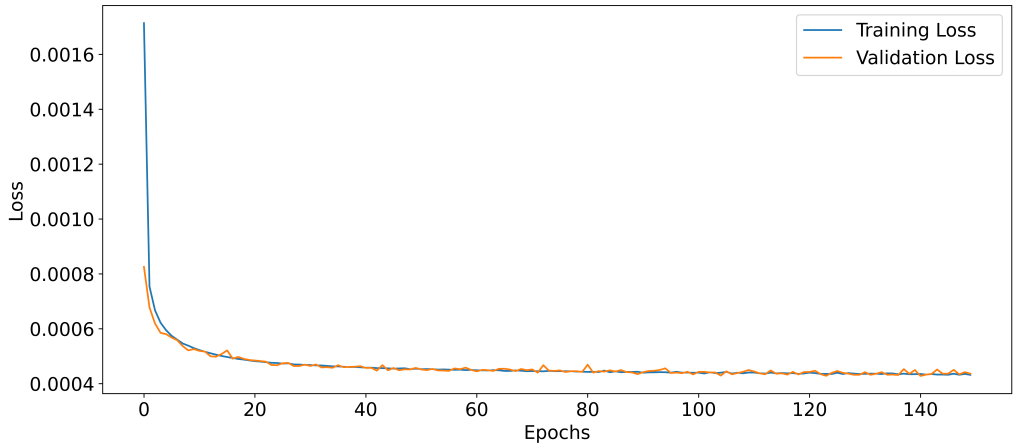


(b) Loss for the complete model of hip flexion

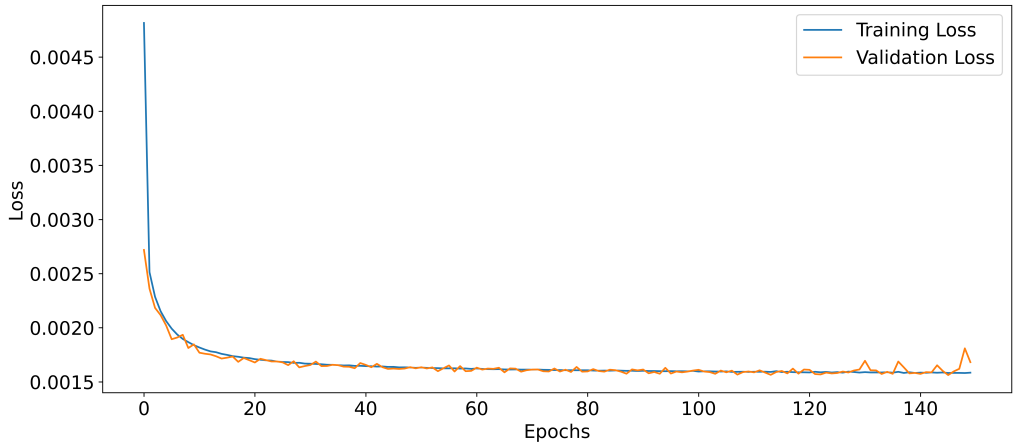


(c) Loss for the complete model for the knee flexion

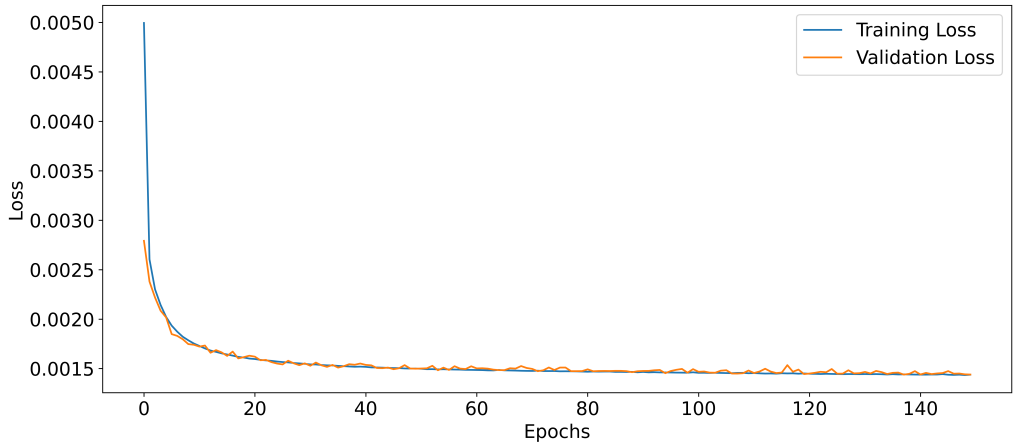
Figure 5.11: Representation of the loss curve for the model fig. 4.3 of the 3 DoFs. With the entire population, we obtain these Loss results based on the MSE during the training phase. For fig. 5.11a, we have a final loss of 0.66. fig. 5.11b, the final loss is about 3.6. fig. 5.8c, the final loss is about 3.5.



(a) Loss for the modified complete model of the hip abduction

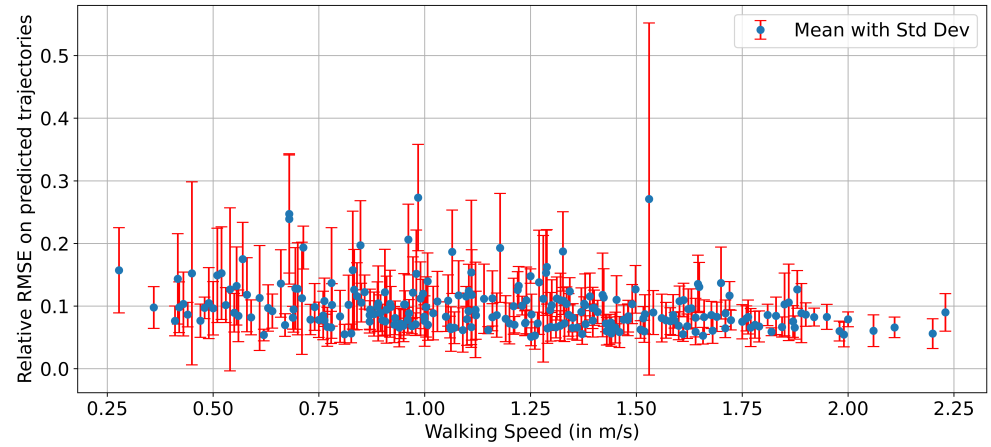


(b) Loss for the modified complete model of hip flexion

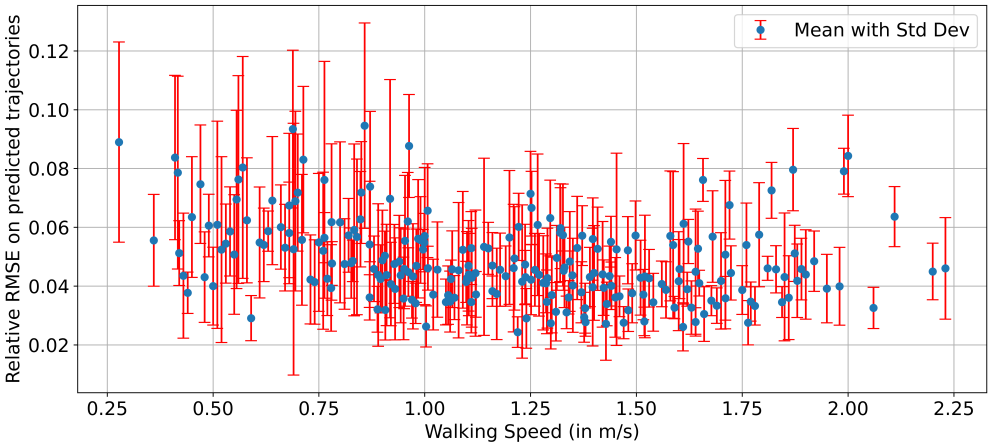


(c) Loss for the modified complete model for the knee flexion

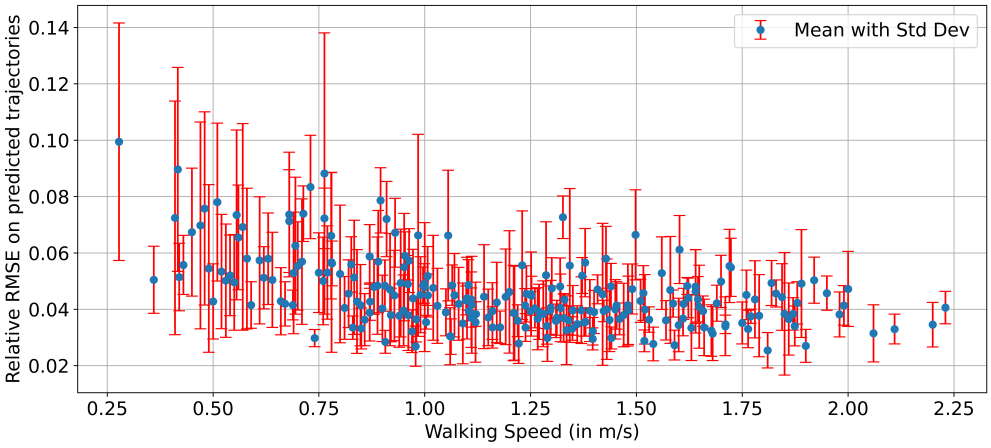
Figure 5.17: Representation of the loss curve for the modified complete model fig. 4.3 of the 3 DoFs. With the entire population, we obtain these Loss results based on the MSE during the training phase..



(a) Error on predicted trajectories for hip abduction

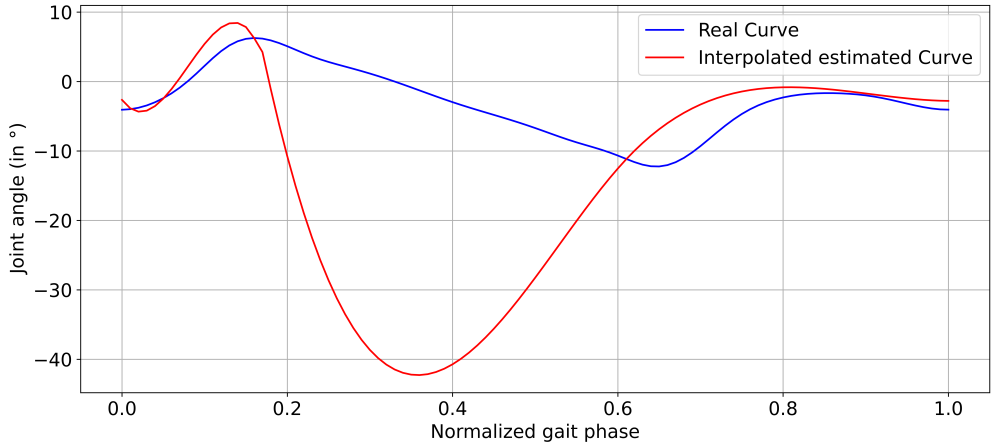


(b) Error on predicted trajectories for hip flexion

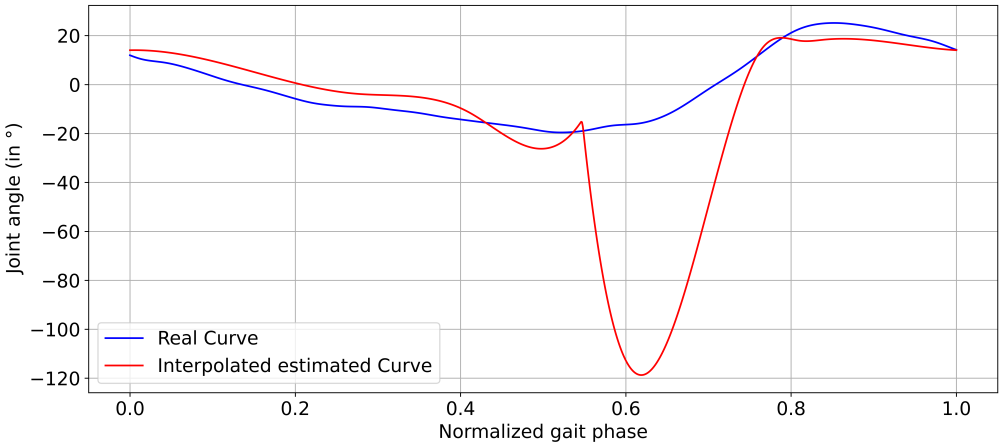


(c) Error predicted trajectories for knee flexion

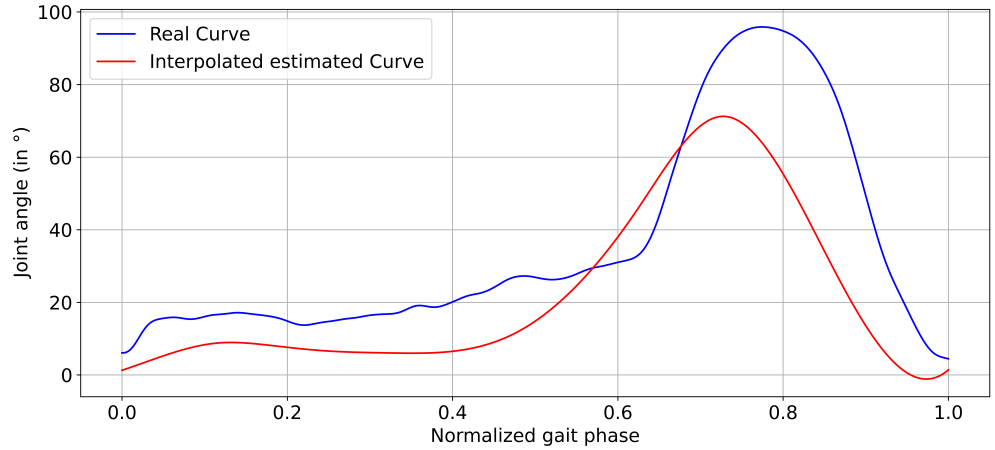
Figure 5.18: Representation of the relative RMSE of the predicted trajectories with the modified complete model fig. 4.3 as function of the walking speed. Each of the figue represent for a DoF, give a specific walking speed, the mean and standard deviation of the error made on the angular position when predicting the key points. The objective is to observe the worse cases scenario for this errors.



(a) Worst predicted trajectory for Hip abduction

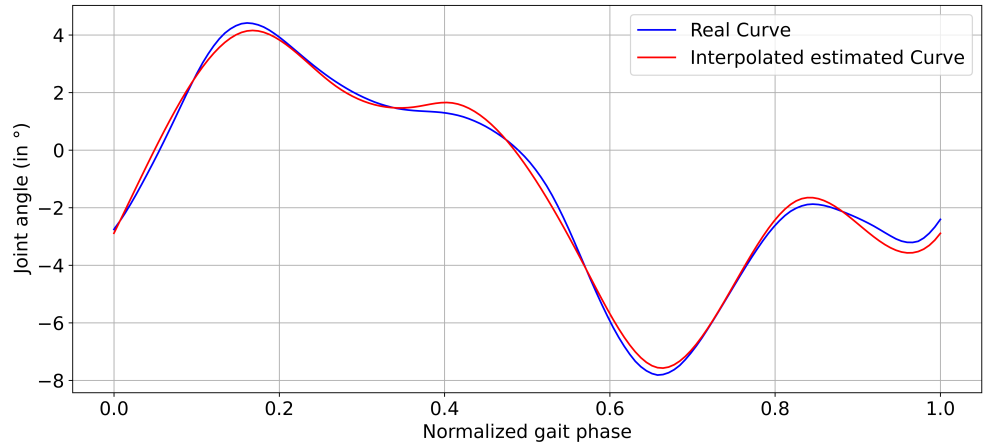


(b) Worst predicted trajectory for Hip flexion

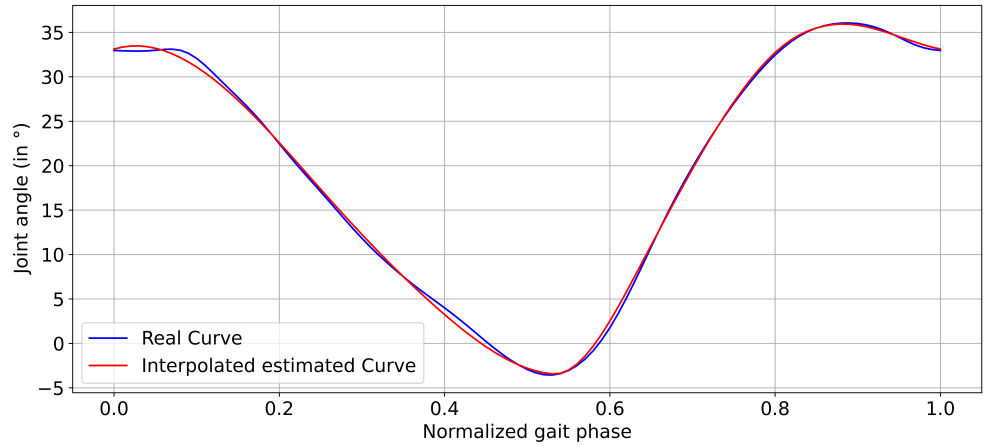


(c) Worst predicted trajectory for Knee flexion

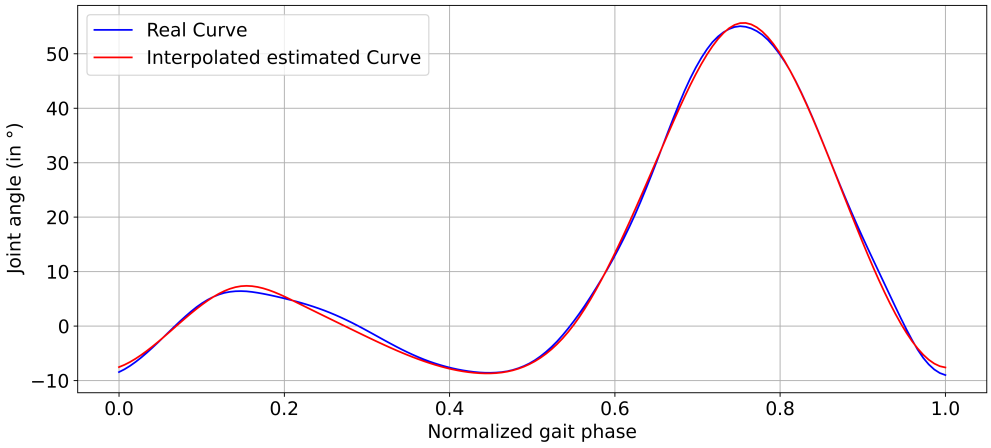
Figure 5.19: Representation of the worst predicted trajectories with the modified complete model for the 3 DoF fig. 4.3. The relative RMSE for this worst prediction are: 0.21 for the knee flexion, 0.72 for the hip flexion and 1.04 for the hip abduction.



(a) Best predicted trajectory for Hip abduction



(b) Best predicted trajectory for Hip flexion



(c) Best predicted trajectory for Knee flexion

Figure 5.20: Representation of the best predicted trajectories with the complete model for the 3 DoF fig. 4.3. The relative RMSE for this best prediction are: 0.011 for the knee flexion, 0.01 for the hip flexion and 0.019 for the hip abduction.

Chapter 6

Summary

After the work that has been done for this project, it is necessary to draw the appropriate conclusions.

6.1 Discussion

The different components of the controller produce quite varied results. The estimation of the gait period works the best, with a maximum error of 50 ms. The estimation of the gait period requires further investigation to find solutions for boundary errors or to test other types of machine learning models for this estimation.

The central part of this controller, the trajectory generation, shows results aligned with those in the literature, potentially even better than the work it is most inspired by, that of Moissenet [44]. Indeed, the model obtained after the latest modifications provides generally satisfactory results, with only a small percentage of results (<10%) having errors greater than 10% of the actual curve amplitude. The error in trajectory generation is on average 5% across the DoFs, indicating an average RMSE of 3° for knee flexion, 2.5° for hip flexion, and 0.6° for hip flexion. These results are in line with the literature and slightly better than those of Moissenet [44]. Additionally, the ease of varying subjects seems to be achieved since it is sufficient to modify the subject's biomechanical parameters as input.

However, for real-time applications, further developments are needed. Trajectories with relative RMSEs greater than 10% can pose a danger to the exoskeleton user. Furthermore, complete controller tests with all its components working in tandem remain to be done, whether offline or on the exoskeleton.

6.2 Conclusion

The field of robotics, and exoskeletons in particular, has been booming in recent years. Exoskeletons are being used in a wide variety of fields: in industry, to alleviate fatigue and limit injuries, for haptic interaction in VR, and for rehabilitation and assistance for the disabled. The field has become so popular that competitions such as the Cybathlon exist to shed light on the integration of disabled people into our society, and to bring together students, engineers and researchers from all walks of life to discuss exoskeletons to help these people. This also leads to the creation of student initiatives such as TUM DASH, whose aim is to create their own exoskeleton. To do this, they need to develop a sophisticated control framework.

For this control framework, there is a typical hierarchy with three levels of controllers: high level, which estimates the user's intention, mid level, which translates this intention

into a trajectory to drive the actuators, and low level, which ensures that the trajectories are precisely followed by the actuators. These controllers can also be categorized according to their structure, resulting in model-based controllers and physical parameters-based controllers.

In this work, the objective was to create a high-level controller capable of generating the trajectories for 3 DoFs (Knee flexion, Hip flexion, and Hip abduction) over the course of a gait cycle, in the context of walking on a flat surface. To achieve this, the foundational components of the controller were built using Machine Learning models, particularly Gaussian Process Regression (GPR) and Neural Networks. The concept was to identify key points on the trajectories and generate these key points using a neural network, with the current joint position and the corresponding gait percentage as inputs, along with the pilot's biomechanical parameters (age, gender, height, weight). This neural network is complemented by a GPR to estimate the pilot's current percentage of the gait cycle, a GPR to estimate the gait period, and a spline interpolation method to reconstruct the curve from the key points. The results obtained seem promising for future development.

6.3 Outlook

Although it is functional for a start, the controller has areas for improvement. It is possible to expand the database to include more subjects of varying sizes, weights, and ages. The most important aspect of the database would be to determine additional filtering criteria. Here, the only criterion was the presence of potential candidates for key points. Another approach would be to remove trajectories where the error between the actual trajectory and the interpolated trajectory is too large.

Regarding interpolation, other methods can be explored. Specifically, the chosen key points impose certain conditions on the curve, such as the first derivative being zero at the extremum and the second derivative being zero at the inflection points. Using a curve reconstruction method that takes these constraints into account could lead to better results. It is also possible to increase the number of key points.

The neural network would require more optimization time to try different hyperparameter values. Additionally, to better capture the non-linearity of the relationships between the data, concepts such as Wavelet Neural Networks (Wavelets NN), which use wavelets to better capture non-linear relationships, can be employed.

The area that requires the most improvement is the estimation of the gait percentage. Using a GPR is very resource-intensive during training. It would be possible to use a neural network, or to stay on the same path, applying the method from [1] to avoid boundary errors is also an idea worth exploring.

Bibliography

[1] A. Seyfarth, Simon Barnikol, Jan Peters, Tag der Einreichung, and Erklärung zur Master-Thesis. “Machine Learning for Active Gait Support with a Powered Ankle Prosthesis”. In: (2014). URL: https://www.semanticscholar.org/paper/Machine-Learning-for-Active-Gait-Support-with-a-Seyfarth-Barnikol/c931d476c3a0e7b429104eff2877da607dceb08a.

[2] Aguirre-Ollinger, G., Colgate, J. E., Peshkin, M. A., and Goswami, A. “Design of an active one-degree-of-freedom lower-limb exoskeleton with inertia compensation”. In: *The International Journal of Robotics Research* 30.4 (2011), pp. 486–499. ISSN: 0278-3649. DOI: 10.1177/0278364910385730.

[3] Anam, K. and Al-Jumaily, A. A. “Active Exoskeleton Control Systems: State of the Art”. In: *Procedia Engineering* 41 (2012), pp. 988–994. ISSN: 18777058. DOI: 10.1016/j.proeng.2012.07.273.

[4] Bengio, Y. “Practical Recommendations for Gradient-Based Training of Deep Architectures”. In: *Neural Networks: Tricks of the Trade* 7700 (2012), pp. 437–478. ISSN: 1611-3349. DOI: 10.1007/978-3-642-35289-8{\textunderscore}26. URL: https://link.springer.com/chapter/10.1007/978-3-642-35289-8_26.

[5] Bottou, L. “Large-Scale Machine Learning with Stochastic Gradient Descent”. In: *Proceedings of COMPSTAT’2010*. Ed. by Lechevallier, Y. and Saporta, G. Dordrecht: Springer, 2010, pp. 177–186. ISBN: 978-3-7908-2604-3. DOI: 10.1007/978-3-7908-2604-3{\textunderscore}16. URL: https://link.springer.com/chapter/10.1007/978-3-7908-2604-3_16.

[6] Bouzbib, E., Teyssier, M., Howard, T., Pacchierotti, C., and Lecuyer, A. “PalmEx: Adding Palmar Force-Feedback for 3D Manipulation with Haptic Exoskeleton Gloves”. In: *IEEE transactions on visualization and computer graphics* PP (2023). DOI: 10.1109/TVCG.2023.3244076.

[7] Chehab, E. F., Andriacchi, T. P., and Favre, J. “Speed, age, sex, and body mass index provide a rigorous basis for comparing the kinematic and kinetic profiles of the lower extremity during walking”. In: *Journal of Biomechanics* 58 (2017), pp. 11–20. ISSN: 0021-9290. DOI: 10.1016/j.jbiomech.2017.04.014. URL: https://www.sciencedirect.com/science/article/pii/S0021929017302130.

[8] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. URL: http://arxiv.org/pdf/1511.07289.

[9] Dey, S., Eslamy, M., Yoshida, T., Ernst, M., Schmalz, T., and Schilling, A. “A Support Vector Regression Approach for Continuous Prediction of Ankle Angle and Moment During Walking: An Implication for Developing a Control Strategy for Active Ankle Prostheses”. In: *IEEE ... International Conference on Rehabilitation Robotics : [proceedings]* 2019 (2019), pp. 727–733. DOI: 10.1109/ICORR.2019.8779445.

[10] Dollar, A. M. and Herr, H. “Lower Extremity Exoskeletons and Active Orthoses: Challenges and State-of-the-Art”. In: *IEEE Transactions on Robotics* 24.1 (2008), pp. 144–158. ISSN: 1552-3098. DOI: 10.1109/TRO.2008.915453.

[11] Eslamy, M. and Alipour, K. “Synergy-Based Gaussian Process Estimation of Ankle Angle and Torque: Conceptualization for High Level Controlling of Active Robotic Foot Prostheses/Orthoses”. In: *Journal of biomechanical engineering* 141.2 (2019). DOI: 10.1115/1.4041767.

[12] Eslamy, M., Oswald, F., and Schilling, A. F. “Estimation of Knee Angles Based on Thigh Motion: A Functional Approach and Implications for High-Level Controlling of Active Prosthetic Knees”. In: *IEEE Control Systems* 40.3 (2020), pp. 49–61. ISSN: 1066-033X. DOI: 10.1109/MCS.2020.2976384.

[13] Eslamy, M., Oswald, F., and Schilling, A. F. “Motion Planning for Active Prosthetic Knees”. In: *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*. IEEE, uuuu-uuuu, pp. 465–470. ISBN: 978-1-7281-5907-2. DOI: 10.1109/BioRob49111.2020.9224433.

[14] Eslamy, M. and Rastgaar, M. “Multi-Joint Leg Moment Estimation During Walking Using Thigh or Shank Angles”. In: *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society* 31 (2023), pp. 1108–1118. DOI: 10.1109/TNSRE.2022.3217680.

[15] Eslamy, M. and Schilling, A. F. “A Conceptual High Level Controller to Walk with Active Foot Prostheses/Orthoses”. In: *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*. IEEE, 8/26/2018 - 8/29/2018, pp. 1224–1229. ISBN: 978-1-5386-8183-1. DOI: 10.1109/BIOROB.2018.8487213.

[16] Farmer, S., Silver-Thorn, S., Voglewede, P., and Beardsley, S. A. “Within-socket myoelectric prediction of continuous ankle kinematics for control of a powered transtibial prosthesis”. In: *Journal of neural engineering* 11.5 (2014), p. 056027. DOI: 10.1088/1741-2560/11/5/056027.

[17] Fukuchi, C. A., Fukuchi, R. K., and Duarte, M. “A public dataset of overground and treadmill walking kinematics and kinetics in healthy individuals”. In: *PeerJ* 6 (2018), e4640. ISSN: 2167-8359. DOI: 10.7717/peerj.4640.

[18] Gal, Y. and Ghahramani, Z. *Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning*. URL: http://arxiv.org/pdf/1506.02142.

[19] Golabchi, A., Riahi, N., Fix, M., Miller, L., Rouhani, H., and Tavakoli, M. “A framework for evaluation and adoption of industrial exoskeletons”. In: *Applied ergonomics* 113 (2023), p. 104103. DOI: 10.1016/j.apergo.2023.104103.

[20] Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., and Schmidhuber, J. “LSTM: A Search Space Odyssey”. In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (2017), pp. 2222–2232. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2016.2582924.

[21] Hastie, T., Friedman, J., and Tisbshirani, R. *The Elements of statistical learning: Data mining, inference, and prediction*. 2nd ed., 12th repr. Springer series in statistics. New York: Springer, 2017. ISBN: 978-0-387-84857-0. DOI: 10.1007/978-0-387-84858-7.

[22] He, K., Zhang, X., Ren, S., and Sun, J. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. URL: http://arxiv.org/pdf/1502.01852.

[23] He, Y., Wu, X., Ma, Y., Cao, W., Li, N., Li, J., and Feng, W. “GC-IGTG: A Rehabilitation Gait Trajectory Generation Algorithm for Lower Extremity Exoskeleton”. In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, uuuu-uuuu, pp. 2031–2036. ISBN: 978-1-7281-6321-5. DOI: 10.1109/ROBIO49542.2019.8961762.

[24] Hinton, G. E. and Salakhutdinov, R. R. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507. ISSN: 1095-9203. DOI: 10.1126/science.1127647.

[25] Huang, G.-B., Wang, D. H., and Lan, Y. "Extreme learning machines: a survey". In: *International Journal of Machine Learning and Cybernetics* 2.2 (2011), pp. 107–122. ISSN: 1868-8071. DOI: 10.1007/s13042-011-0019-y.

[26] Huang, Y., An, H., Ma, H., and Wei, Q. "Modeling and Individualizing Continuous Joint Kinematics Using Gaussian Process Enhanced Fourier Series". In: *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society* 31 (2023), pp. 779–788. DOI: 10.1109/TNSRE.2022.3223992.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Ed. by MIT Press. 2016. URL: https://www.deeplearningbook.org/.

[28] James, G., Hastie, T., Witten, D., and Tibshirani, R. *An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics)*. 2013 Edition. Vol. 103. Springer, 2013.: Springer New York, 2013. ISBN: 978-1-4614-7137-0. DOI: 10.1007/978-1-4614-7138-7.

[29] Jang, J.-S. "ANFIS: adaptive-network-based fuzzy inference system". In: *IEEE Transactions on Systems, Man, and Cybernetics* 23.3 (1993), pp. 665–685. ISSN: 0018-9472. DOI: 10.1109/21.256541.

[30] Jiménez-Fabián, R. and Verlinden, O. "Review of control algorithms for robotic ankle systems in lower-limb orthoses, prostheses, and exoskeletons". In: *Medical Engineering & Physics* 34.4 (2012), pp. 397–408. ISSN: 1350-4533. DOI: 10.1016/j.medengphy.2011.11.018. URL: https://www.sciencedirect.com/science/article/pii/S1350453311003092.

[31] K. Embry, D. Villarreal, R. Macaluso, and R. Gregg. *The Effect of Walking Incline and Speed on Human Leg Kinematics, Kinetics, and EMG*. 2018. DOI: 10.21227/GK32-E868.

[32] Kalita, B., Narayan, J., and Dwivedy, S. K. "Development of Active Lower Limb Robotic-Based Orthosis and Exoskeleton Devices: A Systematic Review". In: *International Journal of Social Robotics* 13.4 (2021), pp. 775–793. ISSN: 1875-4791. DOI: 10.1007/s12369-020-00662-9.

[33] Kazerooni, H., Racine, J.-L., Huang, L., and Steger, R. "On the Control of the Berkeley Lower Extremity Exoskeleton (BLEEX)". In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 4353–4360. ISBN: 0-7803-8914-X. DOI: 10.1109/ROBOT.2005.1570790.

[34] Kiguchi, K., Tanaka, T., and Fukuda, T. "Neuro-Fuzzy Control of a Robotic Exoskeleton With EMG Signals". In: *IEEE Transactions on Fuzzy Systems* 12.4 (2004), pp. 481–490. ISSN: 1063-6706. DOI: 10.1109/TFUZZ.2004.832525.

[35] Kingma, D. P. and Ba, J. *Adam: A Method for Stochastic Optimization*. URL: http://arxiv.org/pdf/1412.6980v9.

[36] Koller, J. R., David Remy, C., and Ferris, D. P. "Comparing neural control and mechanically intrinsic control of powered ankle exoskeletons". In: *IEEE ... International Conference on Rehabilitation Robotics : [proceedings]* 2017 (2017), pp. 294–299. DOI: 10.1109/ICORR.2017.8009262.

[37] Koopman, B., van Asseldonk, E. H. F., and van der Kooij, H. “Speed-dependent reference joint trajectory generation for robotic gait support”. In: *Journal of Biomechanics* 47.6 (2014), pp. 1447–1458. ISSN: 0021-9290. DOI: 10.1016/j.jbiomech.2014.01.037. URL: https://www.sciencedirect.com/science/article/pii/S0021929014000682.

[38] LeCun, Y., Bengio, Y., and Hinton, G. “Deep learning”. In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539. URL: https://www.nature.com/articles/nature14539.

[39] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade* 7700 (2012), pp. 9–48. ISSN: 1611-3349. DOI: 10.1007/978-3-642-35289-8{\textunderscore}3. URL: https://link.springer.com/chapter/10.1007/978-3-642-35289-8_3.

[40] LEMOYNE, R. M. T., ed. *WEARABLE AND WIRELESS SYSTEMS FOR HEALTHCARE I _. (AND FURTHER) EDITION(S): Gait and reflex... response quantification*. Smart Sensors, Measurement and Instrumentation. [S.l.]: Springer Nature, 2024. ISBN: 978-981-97-2438-3. DOI: 10.1007/978-981-97-2439-0.

[41] Liang, F.-Y., Zhong, C.-H., Zhao, X., Lo Castro, D., Chen, B., Gao, F., and Liao, W.-H. “Online Adaptive and LSTM-Based Trajectory Generation of Lower Limb Exoskeletons for Stroke Rehabilitation”. In: *IEEE ROBIO 2018*. [Piscataway, New Jersey]: IEEE, 2018, pp. 27–32. ISBN: 978-1-7281-0377-8. DOI: 10.1109/ROBIO.2018.8664778.

[42] Lo, H. S. and Xie, S. Q. “Exoskeleton robots for upper-limb rehabilitation: state of the art and future prospects”. In: *Medical Engineering & Physics* 34.3 (2012), pp. 261–268. ISSN: 1350-4533. DOI: 10.1016/j.medengphy.2011.10.004. URL: https://www.sciencedirect.com/science/article/pii/S1350453311002694.

[43] Marc G. Genton. “Classes of Kernels for Machine Learning: A Statistics Perspective”. In: *ournal of Machine Learning Research 2* 1 (2001). DOI: 10.1162/15324430260185646.

[44] Moissenet, F., Leboeuf, F., and Armand, S. “Lower limb sagittal gait kinematics can be predicted based on walking speed, gender, age and BMI”. In: *Scientific reports* 9.1 (2019), p. 9510. DOI: 10.1038/s41598-019-45397-4.

[45] Moreira, L., Figueiredo, J., Fonseca, P., Vilas-Boas, J. P., and Santos, C. P. “Lower limb kinematic, kinetic, and EMG data from young healthy humans during walking at controlled speeds”. In: *Scientific Data* 8.1 (2021), p. 103. ISSN: 2052-4463. DOI: 10.1038/s41597-021-00881-3. URL: https://www.nature.com/articles/s41597-021-00881-3.

[46] Nwankpa, C., Ijomah, W., Gachagan, A., and Marshall, S. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. URL: http://arxiv.org/pdf/1811.03378v1.

[47] Pedro Sá Cunha, João Ferreira, A. Paulo Coimbra. “Computational Intelligence generation of subject-specific knee and hip healthy joint angles reference curves”. In: ().

[48] Qian, N. “On the momentum term in gradient descent learning algorithms”. In: *Neural networks : the official journal of the International Neural Network Society* 12.1 (1999), pp. 145–151. DOI: 10.1016/S0893-6080(98)00116-6. URL: https://www.sciencedirect.com/science/article/pii/S0893608098001166.

[49] Rahman, M. H., Kiguchi, K., Rahman, M. M., and Sasaki, M. “Robotic Exoskeleton for Rehabilitation and Motion Assist”. In: *ICIIS 2006*. [Piscataway, NJ]: IEEE, 2006, pp. 241–246. ISBN: 1-4244-0322-7. DOI: 10.1109/ICIIS.2006.365731.

[50] Ramachandran, P., Zoph, B., and Le V, Q. *Searching for Activation Functions*. URL: http://arxiv.org/pdf/1710.05941v2.

[51] Rasmussen, C. E. “Gaussian Processes in Machine Learning”. In: *Advanced Lectures On Machine Learning*. Ed. by Bousquet, O., Luxburg, U. von, and Rätsch, G. Vol. 3176. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Nature, 2004, pp. 63–71. ISBN: 978-3-540-23122-6. DOI: 10.1007/978-3-540-28650-9{\textunderscore}4.

[52] Rasmussen, C. E. and Williams, C. K. I. *Gaussian process for machine learning*. 3. print. Adaptive computation and machine learning. London, England: The MIT Press, 2006. ISBN: 9780262182539.

[53] Rosen, J., Fuchs, M. B., and Arcan, M. “Performances of hill-type and neural network muscle models-toward a myosignal-based exoskeleton”. In: *Computers and Biomedical Research* 32.5 (1999), pp. 415–439. ISSN: 0010-4809. DOI: 10.1006/cbmr.1999.1524. URL: https://www.sciencedirect.com/science/article/pii/S0010480999915240.

[54] Ruder, S. *An overview of gradient descent optimization algorithms*. URL: http://arxiv.org/pdf/1609.04747v2.

[55] Schmidhuber, J. “Deep learning in neural networks: an overview”. In: *Neural networks : the official journal of the International Neural Network Society* 61 (2015), pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0893608014002135.

[56] Schulz, E., Speekenbrink, M., and Krause, A. “A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions”. In: *Journal of Mathematical Psychology* 85 (2018), pp. 1–16. ISSN: 0022-2496. DOI: 10.1016/j.jmp.2018.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0022249617302158.

[57] Scikit-learn. *scikit-learn: machine learning in Python — scikit-learn 1.5.1 documentation*. 5/08/2024. URL: https://scikit-learn.org/stable/.

[58] Scipy. *SciPy documentation — SciPy v1.14.0 Manual*. 24/06/2024. URL: https://docs.scipy.org/doc/scipy/.

[59] Shorter, K. A., Xia, J., Hsiao-Wecksler, E. T., Durfee, W. K., and Kogler, G. F. “Technologies for Powered Ankle-Foot Orthotic Systems: Possibilities and Challenges”. In: *IEEE/ASME Transactions on Mechatronics* 18.1 (2013), pp. 337–347. ISSN: 1083-4435. DOI: 10.1109/TMECH.2011.2174799.

[60] Singh, H., Gupta, M. M., Meitzler, T., Hou, Z.-G., Garg, K. K., Solo, A. M. G., and Zadeh, L. A. “Real-Life Applications of Fuzzy Logic”. In: *Advances in Fuzzy Systems* 2013 (2013), pp. 1–3. ISSN: 1687-7101. DOI: 10.1155/2013/581879.

[61] Singh, R. E. “Gait Analysis”. In: *MOTION ANALYSIS OF BIOLOGICAL SYSTEMS*. Ed. by Singh, R. E. [S.l.]: Springer, 2024, pp. 125–139. ISBN: 978-3-031-52976-4. DOI: 10.1007/978-3-031-52977-1{\textunderscore}8.

[62] Smola, A. J. and Schölkopf, B. “A tutorial on support vector regression”. In: *Statistics and Computing* 14.3 (2004), pp. 199–222. ISSN: 1573-1375. DOI: 10.1023/B:STCO.0000035301.49549.88. URL: https://link.springer.com/article/10.1023/B:STCO.0000035301.49549.88.

[63] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[64] TensorFlow. *TensorFlow*. 6/03/2024. URL: https://www.tensorflow.org/.

[65] Tibshirani, R., Hastie, T., Witten, D., and James, G. *An Introduction to Statistical Learning: With Applications in R*. New York, NY: Springer, 2021. ISBN: 978-1-0716-1417-4. DOI: 10.1007/978-1-0716-1418-1.

[66] Tucker, M. R., Olivier, J., Pagel, A., Bleuler, H., Bouri, M., Lambercy, O., Del Millán, J. R., Riener, R., Vallery, H., and Gassert, R. "Control strategies for active lower extremity prosthetics and orthotics: a review". In: *Journal of NeuroEngineering and Rehabilitation* 12.1 (2015), p. 1. ISSN: 1743-0003. DOI: 10.1186/1743-0003-12-1. URL: https://jneuroengrehab.biomedcentral.com/articles/10.1186/1743-0003-12-1.

[67] Unluhisarcikli, O., Pietrusinski, M., Weinberg, B., Bonato, P., and Mavroidis, C. "Design and control of a robotic lower extremity exoskeleton for gait rehabilitation". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Ed. by Author, I. S. C. IEEE, uuuu-uuuu, pp. 4893–4898. ISBN: 978-1-61284-456-5. DOI: 10.1109/IROS.2011.6094973.

[68] Werbos, P. J. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. ISSN: 0018-9219. DOI: 10.1109/5.58337.

[69] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Journal of Machine Learning Research* (2010). URL: https://www.researchgate.net/publication/215616968_Understanding_the_difficulty_of_training_deep_feedforward_neural_networks/references.

[70] Yang, X., Lihua, G., Yang, Z., and Gu, W. "Lower Extreme Carrying Exoskeleton Robot Adative Control Using Wavelet Neural Networks". In: *Natural Computation, 2008. ICNC '08. Fourth International Conference on*. IEEE / Institute of Electrical and Electronics Engineers Incorporated, 2008, pp. 399–403. ISBN: 978-0-7695-3304-9. DOI: 10.1109/ICNC.2008.754.

[71] Young, A. J., Kuiken, T. A., and Hargrove, L. J. "Analysis of using EMG and mechanical sensors to enhance intent recognition in powered lower limb prostheses". In: *Journal of neural engineering* 11.5 (2014), p. 056021. DOI: 10.1088/1741-2560/11/5/056021.

[72] Yun, Y., Kim, H.-C., Shin, S. Y., Lee, J., Deshpande, A. D., and Kim, C. "Statistical method for prediction of gait kinematics with Gaussian process regression". In: *Journal of Biomechanics* 47.1 (2014), pp. 186–192. ISSN: 0021-9290. DOI: 10.1016/j.jbiomech.2013.09.032. URL: https://www.sciencedirect.com/science/article/pii/S0021929013004879.

[73] Zeiler, M. D. *ADADELTA: An Adaptive Learning Rate Method*. URL: http://arxiv.org/pdf/1212.5701v1.

[74] Zhang, H., Tay, M. O., Suar, Z., Kurt, M., and Zanotto, D. "Regression Models for Estimating Kinematic Gait Parameters with Instrumented Footwear". In: *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*. IEEE, 8/26/2018 - 8/29/2018, pp. 1169–1174. ISBN: 978-1-5386-8183-1. DOI: 10.1109/BIOROB.2018.8487972.

[75] Zou, H. and Hastie, T. "Regularization and Variable Selection Via the Elastic Net". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 67.2 (2005), pp. 301–320. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2005.00503.x.

Disclaimer

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Garching, August 5, 2024

(Signature)