Università degli Studi di Padova

École normale supérieure Paris-Saclay

Dipartimento di Ingegneria dell'Informazione

Master Degree in ICT for Internet and Multimedia

The Department of Mathematics - Master Degree In MVA

Applying Reinforcement Learning to the Controllers of Exoskeleton Robots to Improve their Flexibility and Smoothness of Movement

Supervisor:
Prof.ssa Federica Battisti
Prof.Olivier BRUNEAU
Prof.Miguel Colom
Prof.Bastien Berret
Lucas Quesada

Candidate:
Ali Ahmadi
UNIPD: 2055576
ENS:2300562

Academic Year 2023-2024

Abstract

Exoskeletons are robotic systems that have emerged as a significant innovation over the past century and are poised to play a critical role in the future. At present, they remain in the research phase, although some medical research laboratories are employing them to assist patients with severe injuries to the upper and lower limbs. Moreover, certain manufacturing facilities are investigating the potential integration of exoskeletons into their operational processes.

One of the most significant challenges in the development of exoskeletons is the achievement of flexibility and humanoid-like movement. The attainment of this degree of fluid motion necessitates the accurate anticipation of human decision-making at each discrete moment in time. Nevertheless, it is not currently feasible to access the brain, which is the core of human decision-making. Even with sophisticated techniques such as electrode implantation, these approaches would be costly, invasive, and impractical, at least based on the current state of knowledge.

The objective of this project was to enhance the flexibility of the exoskeleton's movements by combining reinforcement learning with a current controller of the robot which is a proportional-integral-derivative (PID) controller. This approach was designed to optimize the outputs of the exoskeleton's motors to reduce the jerk effect, and ultimately improve the smoothness of its movements.
Reinforcement learning was employed to identify actions that could minimize the jerk factor, allowing for smoother motion. The model successfully learned which actions would lead to reduced jerk, resulting in more fluid movement. Additionally, we developed a simulated environment that replicates exoskeleton dynamics, enabling us to train and test the reinforcement learning agent before transferring it to the actual exoskeleton robot.

Acknowledgements

I want to thank my family who supported me during all the hard and happy times of my life, especially after my studies in Italy and France, my heart will always be with them and I thank them for their support and love.
I want to thank my friends who become like my second home, every time I felt tired and hopeless they helped me, and truly would not be here without the support of these people who accept me and give me unconditional love, they are great people and you always will be in my heart.
I want to thank Padova and Italy, people with great hearts who support us and open a way for us that we can grow and study in one of the greatest universities in the world and meet the greatest people ever.
I want to thank my supervisor Professor Federica Battisti who helped me, especially during my Erasmus program, and guided me through the time, especially with my thesis, thanks, Professor.
I want to give my gratitude to My supervisors and colleagues at CIAMS Laboratory of Paris Saclay and LURPA Robotic Laboratory of ENS Paris Saclay, MVA program Teachers, and employees of ENS Paris Saclay and also my friends in Paris that we could survive through the tough moments and share the happy time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the field of robotic exoskeleton control, it is critical to accurately predict the intention of the user.[15] The movement of an exoskeleton that interacts with humans is frequently observed to be less than smooth. This is primarily attributable to the delay in the exoskeleton controllers in interpreting human intentions, processing the information, and generating the requisite motor commands. Consequently, the resulting movement is typically less fluid than that of a natural human motion.

What, then, is the solution? One potential solution is to directly access the human nervous system by implanting sensors within the body. However, such systems are invasive, costly, and potentially dangerous. It is therefore proposed that a more feasible approach would be to utilize non-invasive methodologies, such as electromyography (EMG) sensors, which are commonly employed for the measurement of muscle movement. The objective of this project was to address a significant challenge in the field of exoskeletons and humanoid robots, namely how to make their movements smoother, more flexible and more human-like.

One of the primary challenges in this field is the difficulty in predicting human movement in real-time and adjusting the exoskeleton accordingly. In the typical case, signals containing movement data are transmitted from the brain to the muscles via the nerves. The requisite intervention type varies according to the patient's level of disability and the location of the issue. Nevertheless, the real-time prediction of human movement remains a challenging undertaking.

Although the implantation of sensors within the brain or nervous system could facilitate real-time monitoring of movement, this method is invasive, expensive and challenging to implement. A more practical solution, particularly for individuals with partial mobility, is to monitor muscle activity using electromyography (EMG) sensors. The sensors facilitate the capture of the electrical activity of muscles, thereby providing insights into human movement.

In this project, we employed several EMG sensors to monitor the movement of the human arm. Based on the data collected from these sensors, we applied appropriate voltage commands to the motors of the exoskeleton to provide the necessary torque to assist the user in performing tasks. To compute the required torque, we utilized a proportional–integral–derivative (PID) control method, which we will discuss in more detail in Section [PIDs].

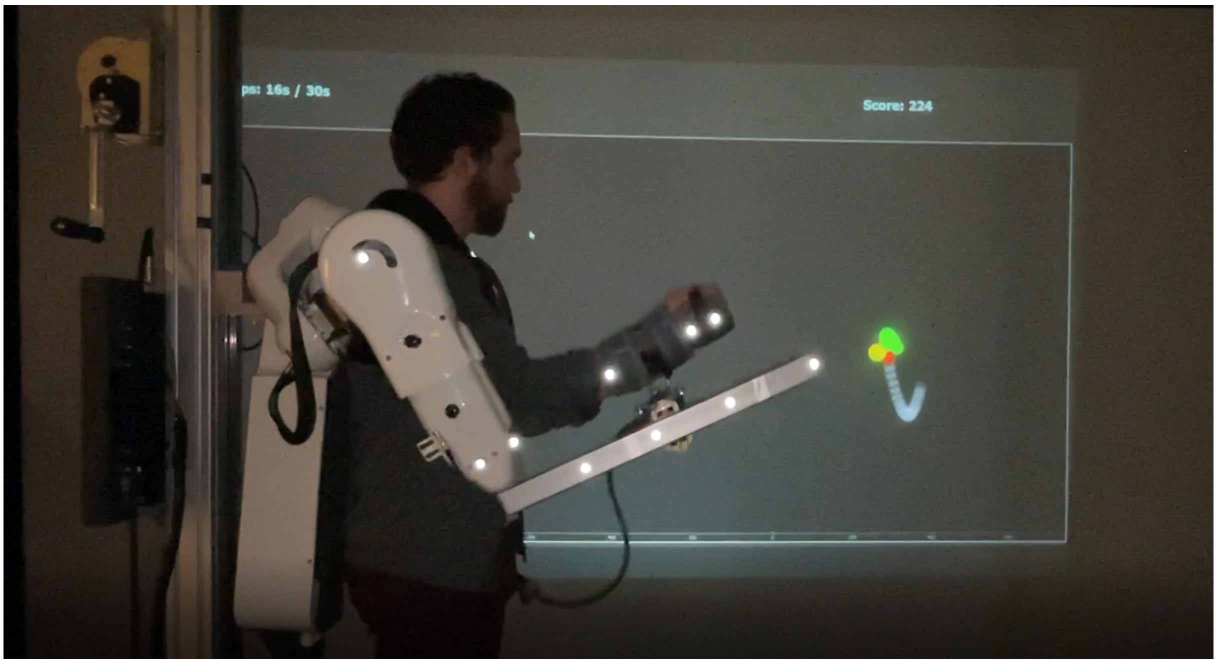

Figure 1.1: One of the PhD students at CIAMS Paris Saclay Laboratory is trying to experiment with the Exoskeleton, in this while he wears Exoskeleton he tries to follow the random point that appears on the monitor.

Chapter 2

Related Works

Lower-body exoskeleton control that adapts to users and provides assistance as needed can increase user participation and motor learning and allow for more effective gait rehabilitation. Adaptive model-based control methods have previously been developed to consider a user's interaction with an exoskeleton; however, the predefined dynamics models required are challenging to define accurately, due to the complex dynamics and nonlinearities of the human exoskeleton interaction. [17] Exoskeletons have gained significant attention in both medical and industrial applications due to their potential to assist individuals with mobility impairments and augment human capabilities in physically demanding tasks. Despite the progress made, one of the key challenges in exoskeleton design remains achieving smooth, flexible, and human-like movements. This challenge is exacerbated by the complex, nonlinear interactions between the exoskeleton and the human body, which are difficult to model accurately using traditional control methods. As a result, recent research has increasingly focused on leveraging reinforcement learning (RL) techniques to address these challenges and improve exoskeleton control systems.

Traditional exoskeleton control methods have often relied on predefined dynamic models to generate control signals. These model-based approaches aim to adapt to a user's specific characteristics, such as walking speed or gait pattern, by utilizing feedback from the exoskeleton-user interaction. For example, model predictive control (MPC) and impedance control have been employed to adjust the stiffness and torque of exoskeleton joints based on user input and feedback[17]. However, accurately defining the dynamics of the human-exoskeleton interaction remains a significant challenge due to the nonlinear and unpredictable nature of these interactions. Simplified models may not capture the full complexity of the system, leading to sub-optimal control performance and reduced user comfort.

To address these limitations, reinforcement learning has emerged as a promising alternative for exoskeleton control. RL techniques enable the system to learn optimal control policies directly from interaction data, eliminating the need for explicit dynamic models. This is particularly advantageous in scenarios where the dynamics are difficult to model or where the system must adapt to varying user behaviors and environments. Recent studies have demonstrated the effectiveness of RL in exoskeleton control by allowing the system to learn from user-specific data and optimize performance over time.

For instance, Rose et al. (2021) introduced a model-free deep reinforcement learning (DRL) approach that allows an exoskeleton to learn and track desired gait patterns without relying on predefined models[17]. Their work focused on the rehabilitation of post-stroke individuals, where the exoskeleton had to adapt to varying gait patterns and user-specific perturbations. By using DRL, the system was able to accurately follow both seen and unseen gait trajectories, demonstrating robustness across different users and rehabilitation stages. This approach not only improves the accuracy of gait tracking but also enhances the overall rehabilitation process by promoting active user participation and motor learning.

Similarly, Utku et al. (2024) applied a deep reinforcement learning framework, specifically the Proximal Policy Optimization (PPO) algorithm, to minimize ground reaction forces (GRF) on crutches during exoskeleton-assisted walking[21]. Their work addressed the often-overlooked issue of upper-body effort in lower-limb exoskeleton users. By minimizing GRF, the system effectively reduced the metabolic energy expenditure of the user, thereby improving comfort and usability. The use of PPO, a state-of-the-art RL algorithm, enabled the exoskeleton to generate joint torques based on real-time feedback from the user's movements, leading to more natural and efficient walking patterns. This study highlights the potential of RL to optimize exoskeleton performance by dynamically adapting to user-specific needs and minimizing undesirable forces.

Another significant contribution to the field is the work by Bazzocchi et al., who explored the use of model-free DRL for end-to-end control of exoskeleton gait patterns[17]. Their approach focuses on generating control policies that can handle the continuous, high-dimensional observation and action spaces required for smooth and adaptive gait control. Unlike traditional RL methods that are limited to discrete action spaces, their DRL approach enables continuous control of joint torques, which is essential for achieving natural and fluid movements in exoskeletons. The ability to generalize across different users and adjust to varying levels of motor functionality further demonstrates the advantages of DRL in personalized rehabilitation.

Furthermore, reinforcement learning has been applied in the context of reducing forces on crutches and optimizing gait patterns to enhance user comfort and reduce fatigue. The study by Utku et al. (2024) provides a clear example of how RL can be used to address specific challenges in exoskeleton control, such as reducing the load on a user's upper body during walking[21]. By using a custom reward function that penalizes high GRF, their approach ensures that the exoskeleton generates smoother and more efficient walking patterns, which is critical for long-term user comfort and adherence to rehabilitation protocols.

These advancements in reinforcement learning-based exoskeleton control are aligned with the goals of this study, which seeks to enhance the smoothness and naturalness of exoskeleton movements. By integrating RL with traditional control methods like PID controllers, this research aims to decrease the jerk factor, a critical metric for assessing movement smoothness. The combined use of RL and PID control allows the exoskeleton to not only learn from user interactions but also maintain a baseline level of stability and responsiveness. This hybrid approach is expected to improve the overall performance of exoskeletons in both rehabilitation and industrial applications, making them more adaptable and user-friendly.

In conclusion, the application of reinforcement learning in exoskeleton control represents a significant step forward in addressing the challenges of smooth and adaptive movement. The ability of RL algorithms to learn from interaction data and optimize control policies in real time makes them particularly suited for complex, human-in-the-loop systems like exoskeletons. As this field continues to evolve, further research is needed to refine these approaches and ensure their scalability and effectiveness in real-world applications. The insights gained from recent studies will undoubtedly contribute to the development of more advanced and capable exoskeleton systems in the future.

Chapter 3

Proposed Method

3.1 Goal

The proposed method is divided into three principal sections: Simulation, Data, and Agent Part. In each section, we will present the methods employed and the fundamental concepts underlying our project. The objective of this project is to enhance the fluidity of the exoskeleton's arm movement while it interacts with and assists the human operator. Before this project, our exoskeleton was controlled by a basic PID controller that attempted to replicate the torque generated by a human operator and apply the PID control formula to generate desired torque commands for the exoskeleton (or voltage for the motor) to facilitate the human arm's movement. Also, the torque of humans has been obtained by a technique discussed in this paper [15], that they took the EMG sensor's data and imported them as an input to a machine learning model and found a relationship between 8 EMG sensors and Torque of the human, which in this case torque of the elbow and shoulder and we used their estimation of torque in our lSTM model.

The objective of our project was to enhance the existing PID method and to attempt to estimate and predict the human movement pattern. This would then allow us to modify the torque command or voltage to the motor in a way that would result in a smoother movement, while also providing the appropriate torque to assist the human. To achieve this, we thought to train our deep reinforcement learning model in a way that would result in a smoother movement, while also providing the appropriate torque for the movement. To do this, we will discuss in more detail the reward function that attempts to achieve a balance between the jerk and PID factors to achieve our goal. One of the innovations that we thought to implement was to train a deep learning model with actual data of the exoskeleton that we collected from previous experiments and build an AI model based on that data to act as an exoskeleton and serve as a simulator for training the reinforcement learning agent. Once a simulator of the exoskeleton had been created, the PPO deep reinforcement learning agent was trained on top of it. The aim was to enable the agent to make decisions that would achieve two specific goals: minimizing the jerk to make the movement smoother and providing the required torque based on PID decisions. The following chapters will discuss the design, construction and experimentation of each model. The first model will simulate the exoskeleton, and the second will train the reinforcement learning agent to predict torque commands for the exoskeleton, thereby making the movement smoother. As illustrated in the figure below, the initial step involves the capture of the subject's current arm movement through the use of EMG sensors, which are strategically positioned on the human arm (eight sensors). This is followed by the application of the aforementioned model. [15] The inverse transformation is then applied to the predicted torque for the elbow and shoulder, which are designated as Th3 and Th4. This data is then used to train the LSTM model, along with the data about the voltages of the exoskeleton motors, which will be discussed in greater detail in the subsequent sections. Subsequently, following the training of our model to function as an environment for reinforcement learning, we train our reinforcement learning agent on this environment. This involves receiving current states as input, which comprise the current human torques and angular velocities from previous actions, as well as the reward for those actions. The agent then outputs decisions or actions, which in this case are voltages for the elbow and shoulder exoskeleton motors. These actions are rewarded positively if they result in a decrease in the jerk factor, which is defined in the reward function and will be discussed in more detail later. The Th3 and Th4 data used for training the LSTM model are derived from offline data collected from previous experiments with exoskeletons. Following the training of the agent, the weights of the agent and model will be saved, and the simulated environment will be removed. The agent will then be tested on real humans and exoskeleton robots. This allows the agent to be tested and fine-tuned in the real world, with any errors identified and subsequently rectified. Following this, the agent will be personalized for each patient, to provide them with a more tailored and comfortable experience when using the exoskeleton.

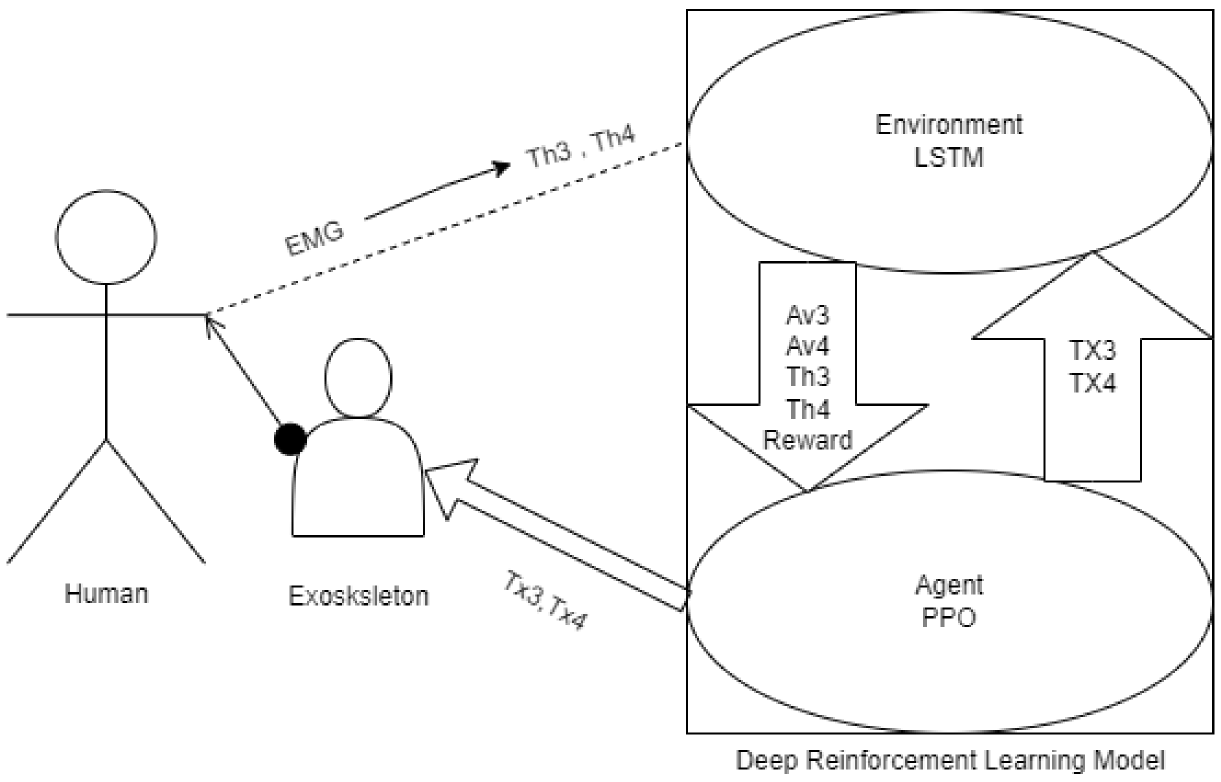

Figure 3.1: Schematic of Human Exoskeleton Interaction, In the right side, the Reinforcement learning Agent and Environment .

3.2 Simulation

One of the novel ideas implemented in this paper was the use of data-driven modeling instead of classical physics-based simulation environments, which are often less accurate when compared to real exoskeletons. Rather than relying on standard simulators, such as OpenAI Gym or MuJoCo, which are based on physical formulas, we utilized data from previous experiments conducted with the exoskeleton. Our approach involved building an AI model based on this real-world data to create a more realistic simulation of the exoskeleton. This method is more accurate, cost-effective, and easier to implement compared to traditional physics-based simulators.

As mentioned earlier, to train our reinforcement learning model, we needed to provide an environment where the agent could interact and reduce its loss function. While directly interacting with the exoskeleton would be the ideal solution, it presents several limitations that prevent us from training our reinforcement learning model directly on the robot.

3.2.0.1 Drawbacks

1-Training in reinforcement learning (RL) requires extensive interaction with the environment and significant amounts of time. Given that multiple participants are involved in testing the robot, this process becomes practically challenging.

2-Additionally, it can pose safety risks. During RL training, the agent makes random decisions as it explores the environment and learns the consequences of its actions in order to update its cost function. Since the exoskeleton is in direct contact with humans, these random actions could potentially be hazardous to the user.

3.2.0.2 Related Simulation Methodes

Multiple simulation environments could be used for our project, such as Open Ai Gym, Mujuco and we will discuss them one by one:

3.2.0.3 OpenAI Gym

OpenAI Gym is a toolkit for reinforcement learning research. It includes a growing collection of benchmark problems that expose a common interface and a website where people can share their results and compare the performance of algorithms. This whitepaper discusses the components of OpenAI Gym and the design decisions that went into the software. OpenAI Gym contains a collection of Environments (POMDPs), which will grow over time. See Figure 1 for examples. At the time of Gym's initial beta release, the following environments were included:

• Classic control and toy text: small-scale tasks from the RL literature.

• Algorithmic: perform computations such as adding multi-digit numbers and reversing sequences. Most of these tasks require memory, and their difficulty can be chosen by varying the sequence length.

• Atari: classic Atari games, with screen images or RAM as input, using the Arcade Learning Environment.

• Board games: Currently, we have included the game of Go on 9x9 and 19x19 boards, where the Pachi engine serves as an opponent.

• 2D and 3D robots: control a robot in simulation. These tasks use the MuJoCo physics engine, which was designed for fast and accurate robot simulation. A few of the tasks are adapted from RLLab. Since the initial release, more environments have been created, including ones based on the open-source physics engine Box2D or the Doom game engine via VizDoom.[8]

3.2.0.4 Mujoco

As robotic hardware becomes more complex and capable, the importance of simulation tools increases. Existing physics engines can be used to test controllers that are already designed. However, they lack the speed, accuracy, and overall feature sets needed to automate the controller design process itself. On the other hand, software packages for automatic controller design (to the extent that they exist) are not integrated with physics engines, effectively limiting them to scenarios where the plant dynamics can be written down explicitly. In the absence of adequate tools, the field continues to rely on manual controller designs – which may be a large part of the reason why present-day robots do not perform as well as one may have hoped given the impressive sensors, actuators, and computing power that are available.[20]

MuJoCo stands for Multi-Joint Dynamics with Contact. It is a general-purpose physics engine that aims to facilitate research and development in robotics, biomechanics, graphics and animation, machine learning, and other areas that demand fast and accurate simulation of articulated structures interacting with their environment.

MuJoCo has a C API and is intended for researchers and developers. The runtime simulation module is tuned to maximize performance and operates on low-level data structures that are preallocated by the built-in XML compiler. The library includes interactive visualization with a native GUI, rendered in OpenGL. MuJoCo further exposes a large number of utility functions for computing physics-related quantities.[2]

3.2.1 why did we say No to the available environments and what is our proposed solution?

Despite the advantages of simulation environments, they presented several challenges for our project. For instance, MuJoCo requires specific physical formulas, which were difficult to obtain for our exoskeleton model. Additionally, we did not have sufficient time to build a digital twin of our exoskeleton in this environment. As a result, we switched to OpenAI Gym, which is a robust environment for training reinforcement learning algorithms. However, it was not an ideal fit for our project's requirements. Therefore, we modified the OpenAI Gym environment to better suit our needs.

Specifically, we needed the modified environment to simulate the exoskeleton's behavior, such that it could accept human movement as input, integrate it with the robot's movement, and provide output data on the exoskeleton's position and angular velocity at each time step.

As previously mentioned, our goal was to create a digital copy of our exoskeleton in the software to train our reinforcement learning model. However, direct access to the physical exoskeleton was limited, and the available simulation environments were inadequate for our purposes. Consequently, we turned to machine learning and deep learning models as a solution. We utilized data from previous experiments conducted by our colleagues in the lab. This dataset provided us with the necessary information to train deep learning models and build an accurate digital copy of the exoskeleton.

3.2.2 Machine and Deep learning models to make Simulator

As previously mentioned, we trained several machine learning and deep learning models to simulate our exoskeleton, referred to as ExoMan. The input to these models consisted of the human torque applied at the elbow and shoulder joints, represented by the columns Th3 and Th4, respectively. Additionally, the current of each motor in the exoskeleton, denoted as Tx3 for the elbow motor and Tx4 for the shoulder motor, was also used as input. We treated this motor current as the torque applied by the exoskeleton to assist in the movement. The output of these models was the angular velocity of the elbow and shoulder joints, represented by Av3 and Av4, respectively.

3.2.2.1 SVR

Support Vector Machines (SVM) are learning machines implementing the structural risk minimization inductive principle to obtain good generalization on a limited number of learning patterns. Structural risk minimization (SRM) involves the simultaneous attempt to minimize the empirical risk and the VC (Vapnik– Chervonenkis) dimension. The theory was originally developed by Vapnik and his co-workers on a basis of a separable bipartition problem at the ATT Bell Laboratories. SVM implements a learning algorithm, useful for recognizing subtle patterns in complex data sets. The algorithm performs discriminative classification learning by example to predict the classifications of previously unseen data. Instead of minimizing the observed training error, Support Vector Regression (SVR) attempts to minimize the generalization error bound to achieve generalized performance. The idea of SVR is based on the computation of a linear regression function in a high-dimensional feature space where the input data are mapped via a nonlinear function. SVR has been applied in various fields – time series and financial (noisy and risky) prediction, approximation of complex engineering analyses, convex quadratic programming and choices of loss functions, etc. In this paper, an attempt has been made to review the existing theory, methods, recent developments, and scopes of

SVR. [6]

3.2.2.2 XGBoost

XGBoost is a scalable and efficient implementation of gradient boosting, a technique that combines multiple weak learners (typically decision trees) to form a strong learner. The algorithm works by iteratively adding trees that correct the errors of the previous ones. XGBoost introduces regularization to prevent overfitting and offers features such as tree pruning, parallelization, and handling of missing data. The combination of these features makes XGBoost highly effective for structured/tabular data.

Applications:

Tabular data prediction tasks (e.g., Kaggle competitions), Classification and regression tasks (e.g., customer churn prediction), Feature importance ranking (e.g., determining which factors most affect an outcome)[9]

3.2.2.3 Random Forest

Random Forests are ensemble learning methods that build multiple decision trees during training. Each tree is trained on a random subset of the data and selects random features to make splits, which helps reduce variance and prevent overfitting. During prediction, the outputs of all the trees are combined (averaged for regression or majority-voted for classification) to produce the final output. The randomness in both data selection and feature selection makes Random Forests robust and less prone to overfitting compared to individual decision trees.

Applications:

Classification (e.g., spam detection, disease diagnosis), Regression (e.g., predicting house prices), Feature selection (e.g., identifying the most important features in a dataset)[7]

3.2.2.4 CNN

CNNs are designed specifically for processing grid-like data, such as images. The key idea is to exploit the spatial structure of data through the use of convolutional layers. In a convolutional layer, the network applies a series of filters (small, trainable matrices) across the input image, which helps detect patterns such as edges, textures, and more complex features at higher layers. These layers are followed by activation functions (like ReLU) and pooling layers that reduce the spatial dimensions while retaining important features. Finally, fully connected layers perform the classification task by assigning probabilities to different classes.

Applications:
Image classification (e.g., recognizing objects in images) Object detection and segmentation (e.g., identifying the location of objects in an image) Medical image analysis (e.g., detecting tumors in scans)[12]

3.2.2.5 Recurrent Neural Networks (RNN)

RNNs are a type of neural network where the connections between nodes form a directed cycle, allowing the network to maintain a memory of previous inputs. This makes RNNs well-suited for sequential data, where the current input is related to previous inputs. The network's hidden state is updated at each time step based on both the current input and the previous hidden state, enabling it to capture temporal dependencies. However, standard RNNs can suffer from vanishing gradients, making it difficult to learn long-term dependencies.
Applications:
Natural language processing (e.g., language translation, sentiment analysis) Time series forecasting (e.g., predicting stock prices) Sequence modeling (e.g., speech recognition)[18]
We used one of the famous typical RNN models which is LSTM.

3.2.2.6 Long Short-Term Memory (LSTM)

LSTM is a specialized form of RNN that addresses the vanishing gradient problem by using memory cells and gates. The LSTM cell contains three gates: the input gate, which controls how much of the new input is written to the memory; the forget gate, which decides what information to discard from the memory; and the output gate, which determines how much of the memory is passed to the next time step. This architecture allows LSTMs to learn long-term dependencies and is particularly effective in tasks where the relationship between inputs spans long sequences.
Applications:
Sequence prediction (e.g., predicting sequences of words or events) Machine translation (e.g., converting sentences from one language to another) Speech recognition and synthesis (e.g., voice-to-text systems)[10]

3.3 Data

The dataset we used, part of it you can find it here [14] that has been extracted from some experiments on the Exoskeleton. This dataset includes kinematic, dynamic, and electromyographic data from 17 participants (11 males, age 28.2 ± 7 years, height 175.4 ± 7 cm, weight 70 ± 11 kg). These data were collected during the performance of a sagittal plane upper limb tracking task for single joint (elbow flexion/extension) and multiple joint (elbow and shoulder flexion/extension).[14] Our final dataset consists of 121000 samples of previous experiments that have been carried out with the help of exoskeleton at CIAMS Laboratory at Paris-Saclay University, we gather samples from 6 different experiments of the ES phase of controller which is a type of controlling og the Exoskeleton which you can read more about it at [15], and by concatenating these do some data cleaning we reach to the 121928 sample of data. The dataset consists of several data from EMG sensors that have been gathered through 8 channels of EMG sensors that have been located in different locations of the human arm to measure muscle response or electrical activity in response to a nerve's stimulation of the muscle[1], after receiving the data from EMG sensors we translate them to the human torque as a human elbow torque and human shoulder torque by using methods and model that have been discussed here [15], in our dataset the Torque of human elbow organized as a feature for our Deep learning model with name Th3 and also the human torque shoulder as a Th4. Another important data for us was angular velocity, Angular velocity is gathered in 2 ways, the first one is the angular velocity of the Exoskeleton which has been captured by the velocity sensors that are on the Exoskeleton and the second way is obtaining the Angular velocity of the Human which has been captured by the motion capture system which is the process of capturing the movement of people, animals or objects, transferring it to a 2D or 3D model and animating it with the set of recorded movements. In this way, motion capture aims to produce 3D models that reflect the real movement of moving subjects.[4], in the end, we used the Angular velocity of the Exoskeleton, although the data is almost the same because the arm and Exoskeleton arm are connected and almost the same length of the arm and exoskeleton arm, the motion capture system is an expensive system and might not be available in a real-world system so we decided to use the angular velocity of the Exoskeleton to make it easier for the application that might develop base on this paper. in our dataset, we organized the Angular velocity of the elbow as Av3 and the Angular velocity of the shoulder as Av4. we organized the data in several features as Tx3 and, Tx4 respectively as a Torque of the Exoskeleton we obtained this data by multiplying the constant 11 in the current of each motor of the Exoskeleton, you can find more detail here*** but to have an idea the torque that each motor can produce is proportional to the current times 11.

3.4 Agent: Reinforcement Learning

Reinforcement Learning (RL) is a learning paradigm where an intelligent agent solves sequential decision-making problems through trial and error. The main objective that an RL agent learns to optimize the cumulative return, i.e., a discounted sum of the rewards. This makes the reward a crucial element of the problem, as it defines the optimal decision-making policy that the agent will try to learn.[22] The First question that we want to answer here is why we chose Reinforcement Learning in the Ocean of all the AI different models. Well, Training a deep neural network to maximize a target objective has become the standard recipe for successful machine learning over the last decade. These networks can be optimized with supervised learning if the target objective is differentiable. For many interesting problems, this is however not the case. Common objectives like intersection over union (IoU), bilingual evaluation understudy (BLEU) score, or rewards cannot be optimized with supervised learning. A common workaround is to define differentiable surrogate losses, leading to sub-optimal solutions concerning the actual objective. Reinforcement learning (RL) has emerged as a promising alternative for optimizing deep neural networks to maximize non-differentiable objectives in recent years. Examples include aligning large language models via human feedback, code generation, object detection, or control problems. This makes RL techniques relevant to the larger machine-learning audience. [11] Since we wanted to make the movement of our exoskeleton smoother and more humanoid in real-time, we found Deep Reinforcement Learning Algorithm very useful since we could train our model on the controller of the Exoskeleton by interacting with the Robot in real time this type of model was one of the best solution for our project.

So in the area of Reinforcement Learning, recently we have very good models to test on and Given the complexity and continuous nature of the problem, we found DDPG, TD3, SAC, and PPO as a good candidate.

DDPG and TD3 are well-suited for continuous control tasks, with TD3 offering improvements in stability and performance. SAC provides good stability and exploration, which can be crucial for fine-tuned motor control. PPO is easier to implement and tune and is highly reliable for general continuous control problems normally.

3.4.1 Proximal Policy Optimization Algorithms (PPO)

proximal policy optimization is a family of policy optimization methods that use multiple epochs of stochastic gradient ascent to perform each policy update. These methods have the stability and reliability of trust-region methods but are much simpler to implement, requiring only a few lines of code change to a vanilla policy gradient implementation, applicable in more general settings (for example, when using a joint architecture for the policy and value function), and have better overall performance.[19]

Metrics:

ep_len_mean: This stands for the mean episode length, which is the average number of steps per episode. In this case, it's 300 steps. This indicates how long, on average, the agent interacts with the environment before an episode ends. ep_rew_mean: This is the mean episode reward, which represents the average total reward collected per episode. The value is -6.49e+06, which is a large negative number. A negative reward could indicate that the task is challenging, or it could signal a poorly performing policy, depending on the environment and reward structure.

Time Metrics:

fps: This refers to frames per second, indicating the speed at which the environment is being processed. A low value like 8 fps suggests that either the environment or the training process is computationally expensive.

iterations: This represents the number of iterations or policy updates that have occurred. In this case, there have been 72 iterations so far.

Time_elapsed: This measures the total time that has elapsed during training, in seconds. Here, the model has been training for 2139 seconds (35.65 minutes).

Total_timesteps: This is the cumulative number of timesteps the agent has experienced during training. Here, it's 18,432 timesteps.

Training Metrics:

Approx._kl: KL divergence measures how much the new policy deviates from the old policy after an update. A very low value like 2.6077032e-08 indicates that the policy hasn't changed much between updates, which could suggest conservative updates.

Clip_fraction: This is the fraction of the probability ratios that are clipped during optimization. Since PPO uses clipping to prevent too large policy updates, a clip_fraction of 0 suggests that no clipping occurred, possibly indicating that the updates were within the allowed range.

Clip_range: This is the range within which PPO clips the policy updates to avoid large changes. The value is set to 0.2, which is a common choice.

Entropy_loss: Entropy encourages exploration by ensuring the policy doesn't become too deterministic. A value of -2.84 suggests some degree of randomness in the actions being taken by the policy, but the specific value depends on the environment and policy structure.

Explained_variance: This metric measures how well the value function explains the variance in the rewards. A value of 0 means the value function isn't capturing any of the reward variance, indicating poor performance of the value network.

Learning_rate: The rate at which the model updates its parameters during training. Here, it's set to 0.0003, a standard learning rate for PPO, allowing steady learning without overly aggressive updates.

loss: This is the overall loss of the model, which in this case is 1.38e+11. This includes both policy loss and value loss, and such a large value might indicate a challenging training process, potentially due to high rewards or significant errors in predictions.

n_updates: This indicates the number of updates applied to the model, with 710 updates being performed.

Policy_gradient_loss: This reflects the loss specifically from the policy gradient updates. A very small value like -2.05e-05 suggests minimal updates were made to the policy during this step.

std: This refers to the standard deviation of the action distribution. A value of 1 indicates the amount of exploration in the action space, and it's typical for it to be initialized at 1 and adjusted during training.

Value_loss: This represents the loss associated with the value function. A high value like 2.68e+11 suggests that the value function is struggling to accurately predict the returns, possibly due to high reward variance or complex dynamics.[19][3]
[13]

Hyperparameters in PPO:
The performance of PPO is heavily influenced by several hyperparameters, some of which are seen in your log, while others are more commonly adjusted during training:

1-Learning Rate (learning_rate): Controls the step size during gradient descent. Common values range from 0.0001 to 0.003.

2-Clip Range (clip_range): Determines how much the policy is allowed to change in each update. Typically set between 0.1 to 0.3.

3-Entropy Coefficient: Balances the trade-off between exploration and exploitation by adjusting how much randomness is encouraged in the policy.

4-Discount Factor (gamma): The discount factor for future rewards. A typical value is 0.99, meaning future rewards are slightly less valuable than immediate rewards.

5-GAE Lambda (gae_lambda): Used in Generalized Advantage Estimation to balance bias and variance in the advantage function. A value of 0.95 is commonly used.

6-Batch Size: The number of samples used in each update. Larger batch sizes provide more stable updates but require more computation.

7-Number of Epochs: The number of passes through the data during each update step. Increasing the number of epochs can lead to better training, but also risks overfitting.

8-Number of Timesteps: The total number of Timesteps used during training. More timesteps generally lead to better performance, assuming the model isn't overfitting.

9-Value Function Coefficient: Determines how much the value loss contributes to the total loss. This helps in balancing policy and value network training.

10-Max Grad Norm: Clipping gradients to prevent exploding gradients during training.[19][3] [13]

3.5 Reward Function

Reward function is one of the important factors in training reinforcement learning agents since it is like a guideline to help the agent learn, we tried some different formulas but all of them were based on Jerk and PID output, in the end, we only used the jerk factor and added the PID term into the self-state term.

3.5.1 Jerk

Measuring the quality of movement is a need and a challenge for clinicians. Jerk, defined as the quantity of acceleration variation is a kinematic parameter used to assess the the smoothness of movement. In the field of robotic motion control and biomechanics, the concept of jerk plays a crucial role in determining the smoothness of a movement. Jerk, defined as the rate of change of acceleration, is the third derivative of position concerning time. High levels of jerk are often associated with abrupt changes in motion, which can lead to increased mechanical stress, vibration, and discomfort in systems ranging from industrial robots to human biomechanics.

By minimizing the jerk factor, the resulting motion becomes smoother and more gradual, reducing the wear and tear on mechanical components and decreasing the potential for injury in biological systems. This smoothness is especially important in applications where precision and gentle handling are required, such as in surgical robots, prosthetics, and precision manufacturing. The mathematical representation of jerk is given by:

$$\mathbf{J}(t) = \frac{d^3\mathbf{x}(t)}{dt^3} \tag{3.1}$$

where x(t) represents the position as a function of time. Controlling and reducing jerk can therefore significantly enhance the performance and longevity of both mechanical and biological systems by ensuring that movements are executed smoothly, with minimal abrupt changes in acceleration."

This expanded explanation provides a broader context for the importance of controlling jerks in various applications, tying it back to the benefits mentioned in the paper.[16]

3.5.2 PID

WITH its three-term functionality covering treatment to both transient and steady-state responses, proportional-integral-derivative (PID) control offers the simplest and yet the most efficient solution to many real-world control problems. Since the invention of PID control in 1910 (largely owning to Elmer Sperry's ship autopilot), and the Ziegler–Nichols' (Z-N) straightforward tuning methods in 1942 [5] , the popularity of PID control has grown tremendously. With advances in digital technology, the science of automatic control now offers a wide spectrum of choices for control schemes. However, more than 90 industrial controllers are still implemented based on PID algorithms, particularly at the lowest levels [5], as no other controllers match the simplicity, clear functionality, applicability, and ease of use offered by the PID controller [5]. The proportional term provides an overall control action proportional to the error signal through the all-pass gain factor. The integral term reduces steady-state errors through low-frequency compensation by an integrator. The derivative term improves transient response through high-frequency compensation by a differentiator. A standard PID controller is also known as the "three-term" controller, whose transfer function is generally written in the[5] :

u(t) = K$_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$

$C(s) = K_p + \frac{K_i}{s} + K_d \cdot s$

$C(s) or u(t) is the control output.$

$Kp is the proportional gain.$

$Ki is the integral gain.$

$Kd is the derivative gain.$

$e(t) is the error signal.$

Chapter 4

Experimental Results

4.1 Environment Simulator

So as we mentioned in the Proposed Method parts, we tried to build a simulating environment by using artificial intelligence models, we divided our dataset into the training and tests and fed them to several different machine and deep learning models to build a good simulator environment or digital twin of our exoskeleton.

4.1.1 Preparing Data

we applied different data cleaning methods to handle missing values, deleting unnecessary columns, concatenating data from several experiments, and organizing them in 4 features that are Tx3, Tx4, Th3, Th4, and two outputs as an Av3 and Av4 which you can find the detail of each data at 3.2 Data Part. Normalization: MinMaxScaler is used to scale the features and labels to a range of(1,1).

4.1.2 Support Vector Machines (SVR)

In this model, we used a Support Vector Machine for regression tasks. We trained Two SVR models separately for each target variable (Av3 and Av4). Then the model Predictions from both models are combined and evaluated using mean squared error.

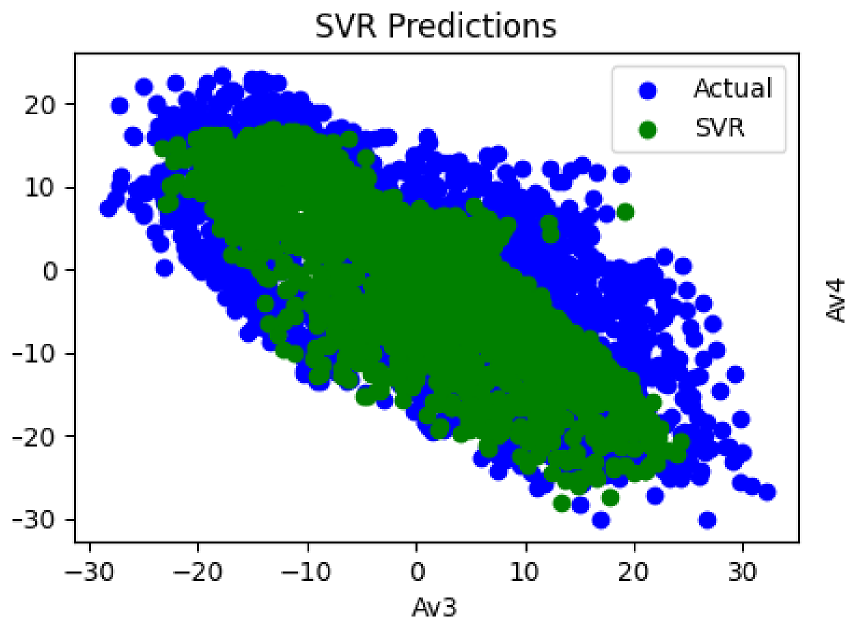

Figure 4.1: This image illustrates the result of Support Vector Machines model prediction, green points are predictions and blue ones are the actual data.

4.1.3 Random Forest

The number of trees is set to 100. The model is trained and evaluated using mean squared error

The model is trained using fit and evaluated using mean squared error.

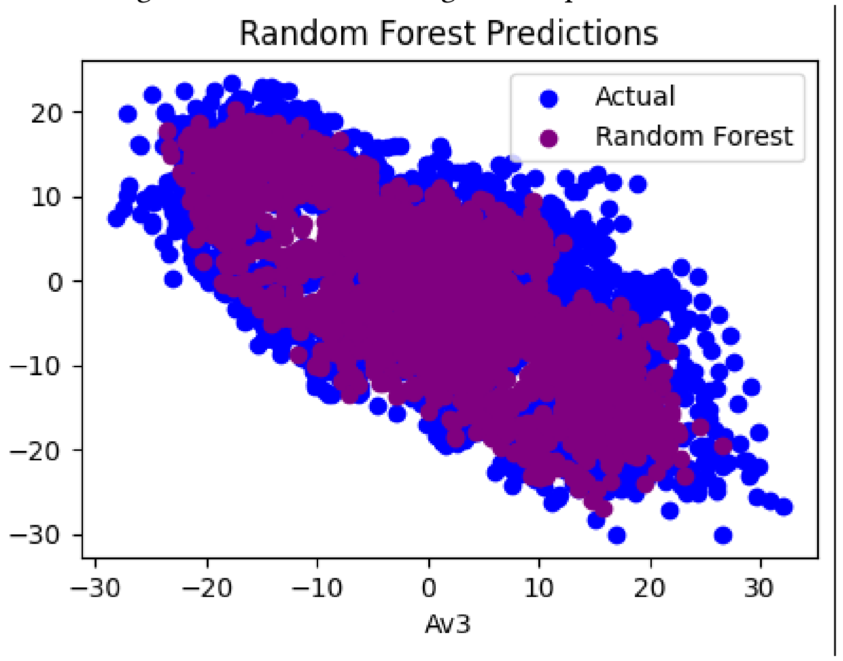

Figure 4.2: This image illustrates the result of the Random Forest model prediction, purple points are predictions and blue ones are the actual data.

4.1.4 XGBOOST

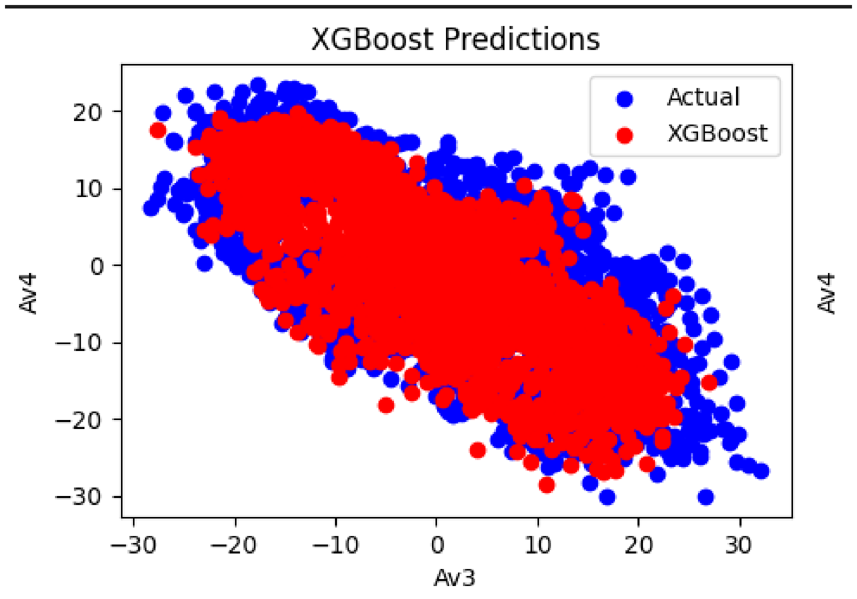

Figure 4.3: This image illustrates the result of the xgboost model prediction, red points are predictions and blue ones are the actual data.

4.1.5 CNN

This model has been designed for sequential data, CNN uses a 1D convolutional layer to capture spatial hierarchies in data. The input shape is adapted for 1D convolution. The network is compiled with the Adam optimizer and Mean Square Error loss. We trained The model for 100 epochs with a batch size of 10.

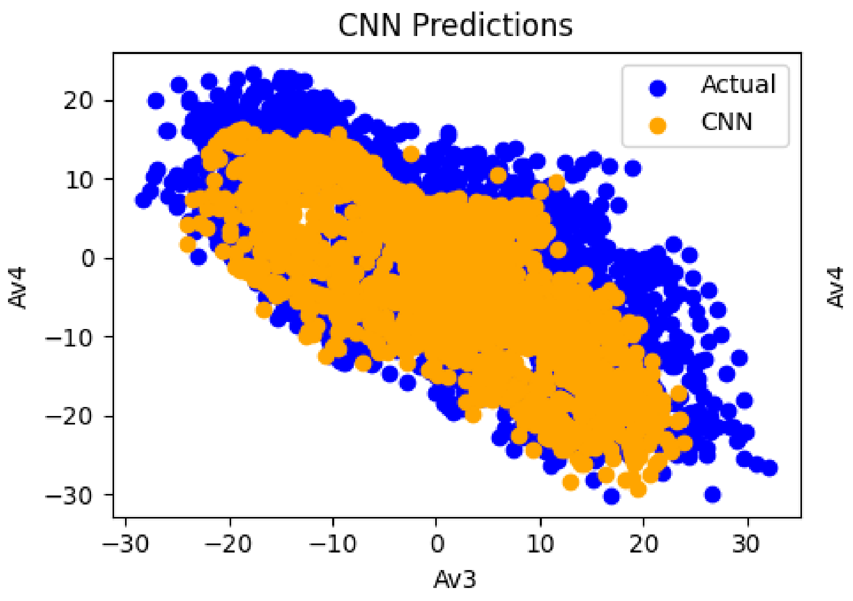

Figure 4.4: This image illustrates the result of the CNN model prediction, yellow points are predictions and blue ones are the actual data.

4.1.6 LSTM (Long short-term memory)

In our LSTM model, we used the TensorFlow Keras sequential model library to build it. It has 2 sequential layers with 50 units each, followed by 2 drop-out layers with a range of 0.2 and, in the end, a dense layer. As you can see in the graph below, the LSTM model predicts with good accuracy based on the 10 previous time steps. The loss function accuracy is 0.004772, which is much better than the previous results of the other models.

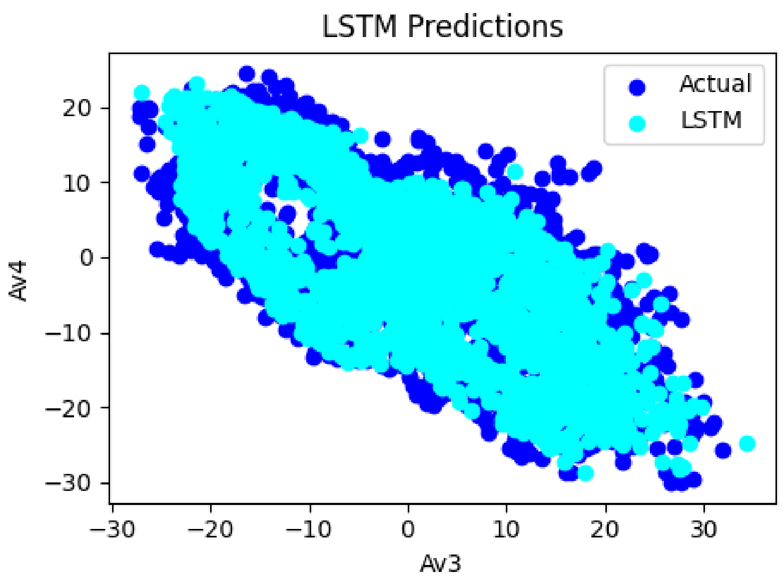

Figure 4.5: This image illustrates the result of the Long short-term memory model prediction based on 10 previous time steps, turquoise blue points are predictions and blue ones are the actual data.

In the below graph, the model predicts based on 100 previous time steps that improved the loss function to 0.0006970633403398097, and as you see the predictions mostly overlap with the actual data.

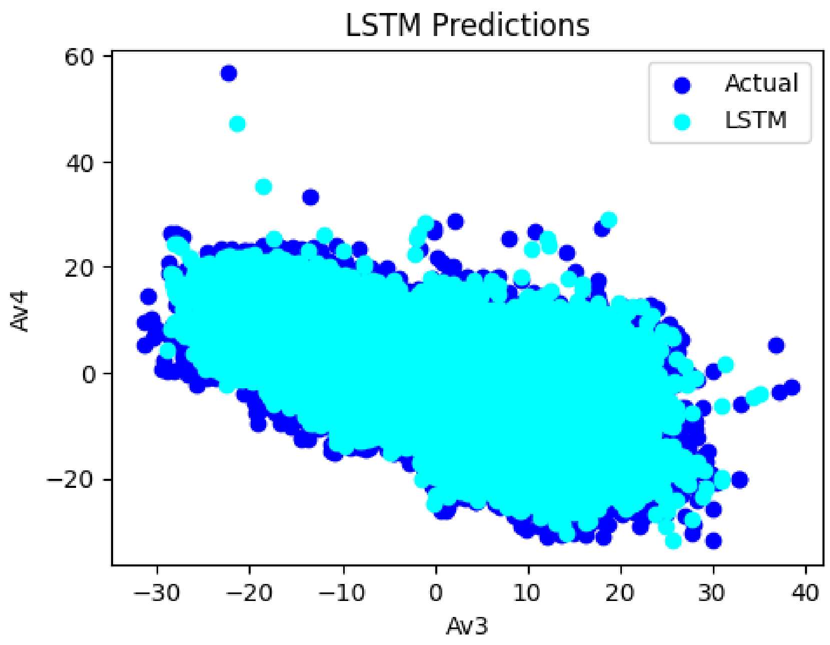

Figure 4.6: This image illustrates the result of the Long short-term memory model prediction based on 100 previous time steps, turquoise blue points are predictions and blue ones are the actual data.

As you can see in the figure, the validation loss is less than the training loss which is due to the dropout layer in our model, as we know from [6] by using drop out layer in our model, the validation loss can be lower than training loss because we do drop some neurons during training that this can cause higher loss during training but in the validation part we test on full neurons so we have better loss than training loss.

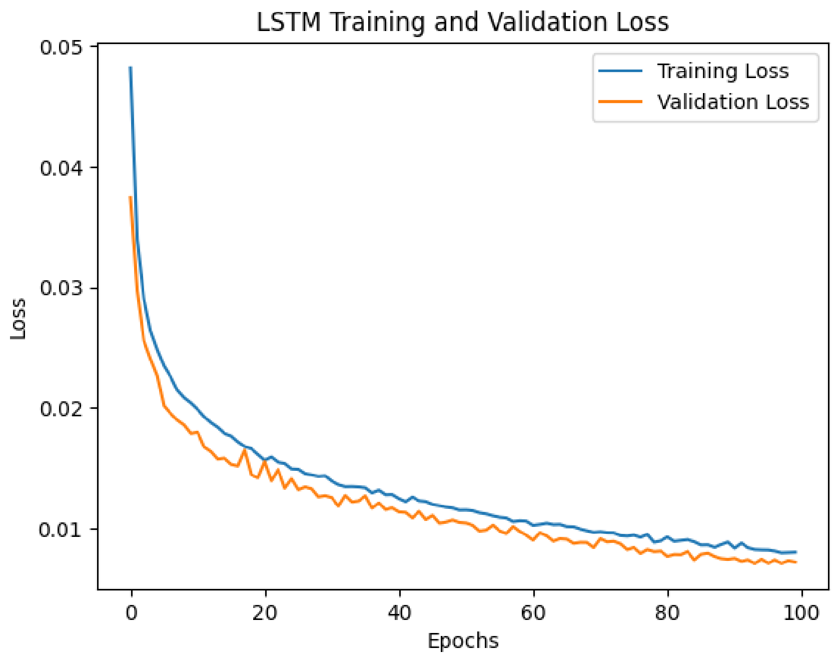

Figure 4.7: LSTM Training and Validation loss, better validation loss is due to regularization techniques specifically drop out .

Loss Function Compare Results		
Model	Loss Function	Rank
LSTM-100	0.0006970633403398097	1
LSTM-10	0.004565627779811621	2
Random Forest	0.011675131523609722	3
XGBoost	0.013300321773123938	4
CNN	0.014365621770928878	5
SVR	0.016126288979828082	6

Table 4.1: Comparison of the different models loss function results.

4.1.7 Compare the Results

As demonstrated by the graphs and the comparison of the normalized loss functions across all tested models, the LSTM model outperformed the others, achieving the best results. We believe this is due to the continuous nature of human and exoskeleton movement, which cannot be accurately captured by evaluating individual time steps in isolation. Instead, it is essential to consider previous movements to better predict subsequent actions.

Consequently, we saved the weights of the trained LSTM model and used it as the simulation model for our exoskeleton. This model effectively serves as the environment for training our reinforcement learning agent. We believe this approach represents a more accurate and innovative method by using a deep learning model, such as LSTM, to simulate real exoskeleton robots and serve as the environment in which reinforcement learning is applied.

4.1.8 Reward Function

One of the important things in the field of Reinforcement learning that affects the behavior of the model so much is the Reward function, we designed several reward functions to get the best performance if you design a bad reward function that can not capture your environment behavior very well or it could converge around a wrong goal or not converge at all. So in our project, we tried several reward functions to find out the best function that could fit with our goal, the first one was minus logarithmic, which gives a positive point every time our distance from the target variable reached 0, and minus logarithmic result to other results. We will discuss the distance from the target variable and the result function in the next section.

$$\text{penalty} = \begin{cases} 100, & \text{if distance\_from\_target} = 0 \\ -\log(\text{distance\_from\_target}), & \text{otherwise} \end{cases}$$

The second reward function that we tried was threshold and square, in way that it gave a high positive score if the distance from the target variable was less than 0.5 and the negative square of the result function distance from the target variable to the agent, you can find the formula of them below:

$$\text{penalty} = \begin{cases} 10, & \text{if distance\_from\_target} < 1 \\ -\text{distance\_from\_target}^2, & \text{otherwise} \end{cases}$$

another reward function that we tried was more complex, it is defined below :

$$\text{penalty} = \begin{cases} -\log(\text{distance\_from\_target}), & \text{distance\_from\_target} < 1 \\ \text{distance\_from\_target}^2, & 2 > \text{distance\_from\_target} > 1 \\ \text{distance\_from\_target}, & 3 > \text{distance\_from\_target} > 2 \\ -\text{distance\_from\_target}, & \text{otherwise} \end{cases}$$

The reason we tried to make our function more complex was to force it to try to minimize the sum of the elbow jerk and shoulder jerk as much as possible, also we should mention that the calculated jerk of the previous experiments that have been done with exoskeleton without our reinforcement learning and only with simple PID controller and the average of Mean shoulder Jerk was: 0.8068689285427039 and the average of elbow jerk was:0.5604592302640197 so we decided to define our loss function in a way that becomes better than our ground truth and we ended up with the above formula.

4.1.9 Result Function

Result Function is the main goal of our project, we want to decrease the Jerk to make the movement smoother and provide proper help to the exoskeleton by making our prediction similar to the output of the PID controller that is implemented to provide the proper torque for the exoskeleton. In other words, we try to minimize the jerk and minimize the error between the output of the PID controller and agent output, below formula is our suggestion for our result function that calculates the error between the angular velocity of the previous time steps and the current angular velocity or in other word jerk and also it calculates the error between PID controller error and agent output which is the current of the motors or by multiplying in a constant 6.3, we can achieve to the torque of the Exoskeleton, so we can calculate the difference between the torque of the exoskeleton that predicted by agent and torque of the exoskeleton that provided by PID formula.

$$\text{Result} = (\text{jerk\_shoulder} + \text{jerk\_elbow})$$

$$\text{Jerk} = (\text{Current Angular Velocity shoulder} - \text{Previous Angular Velocity shoulder})^2$$

4.2 Reinforcement Learning Agent

The decision to utilize PPO was predicated on its robust capabilities and propensity for instability, coupled with its extensive deployment in robotics. A basic PPO agent was employed and subsequently enhanced based on the outcomes of our experiments. Various episode lengths, learning rates, batch sizes (primarily 128, 64, and 32), timesteps, and action spaces were explored. Additionally, alternative loss functions were tested, as discussed in the related section, to facilitate more effective agent learning.

4.2.1 Action Space

Before elucidating the characteristics of our agent, it is imperative to delineate how we delineated the action spaces for our project. Action spaces represent the options we have defined for our agents to make decisions. In this case, our objective is for our agent to select voltages for our two exoskeleton motors (elbow and shoulder) in a manner that minimizes the jerk and can accommodate the required torque was thus defined as a continuous action space between the lowest and highest voltages reached by the motors in previous experiments. However, this action space proved too extensive for the agent to process, resulting in the generation of highly negative and large rewards, as illustrated in Figures 4.8 and 4.9.

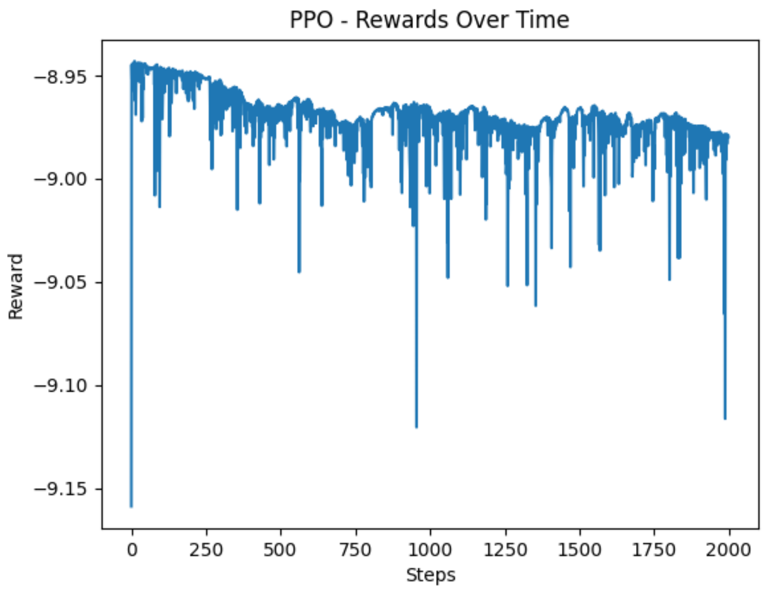

Figure 4.8: This image shows the result of the reward during training of the first tests before improvements

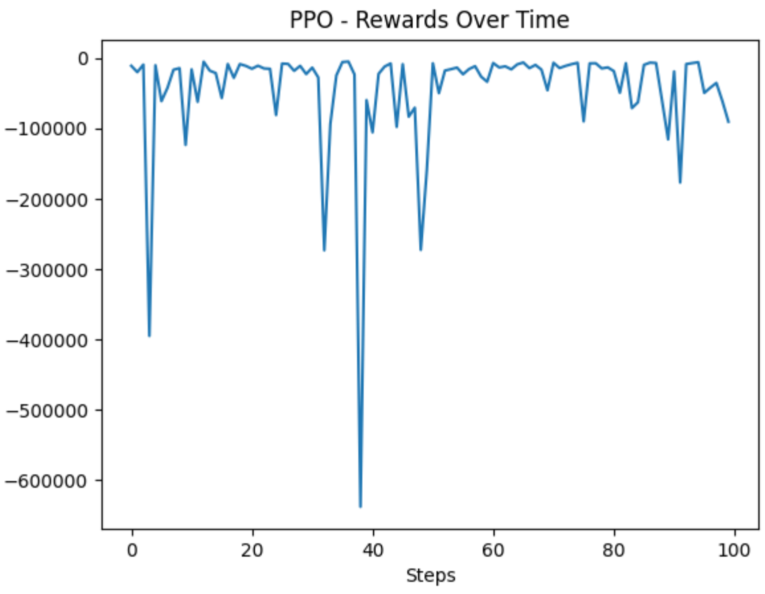

Figure 4.9: This image shows the huge negative reward function due to the first test of the model before improvement in agent and reward functions.

It was thus decided that the action space should be modified to facilitate a more conscious decision-making process. The PID controller output was removed from the reward function and incorporated into the state. Consequently, at each time step, the PID result is returned to the agent as the current state, thereby requiring the agent to make decisions based on the current state. This approach has effectively addressed the issue of a narrow action space for the agent. In this case, the action space is discrete. The agent is instructed to select a number between -1, 0, and 1. If the agent chooses the action 0, it indicates that the PID output is not altered. Alternatively, this can be interpreted as the voltages selected by the PID remaining unchanged. This indicates that the agent has selected 0.1 less voltage than that chosen by PID. The action that increases the PID output by 0.1 is represented by the value 1. Consequently, the action space of the agent has been simplified. In this case, the agent's actions will be confined to a range around the voltage chosen by PID. The objective is to identify which actions within this range will result in a more beneficial reward function. After the aforementioned experimentation, the results are presented in the subsequent image. To ascertain the optimal action space, we conducted trials with higher action spaces, specifically 10 and 100.

4.2.2 PPO (Proximal policy optimization)

We train our PPO reinforcement learning method with different parameters as we discussed and the main hyperparameters ratio is defined below:

learning$_rate = 3e - 4$,
$n_steps = 256$,
$batch_size = 32$,
$n_epochs = 10$,
$gamma = 0.99$,
$clip_range = 0.2$,
$ent_coef = 0.01$,

In the graph below, you can see the result of training our model with an agent with continuous action spaces with 3 choices.

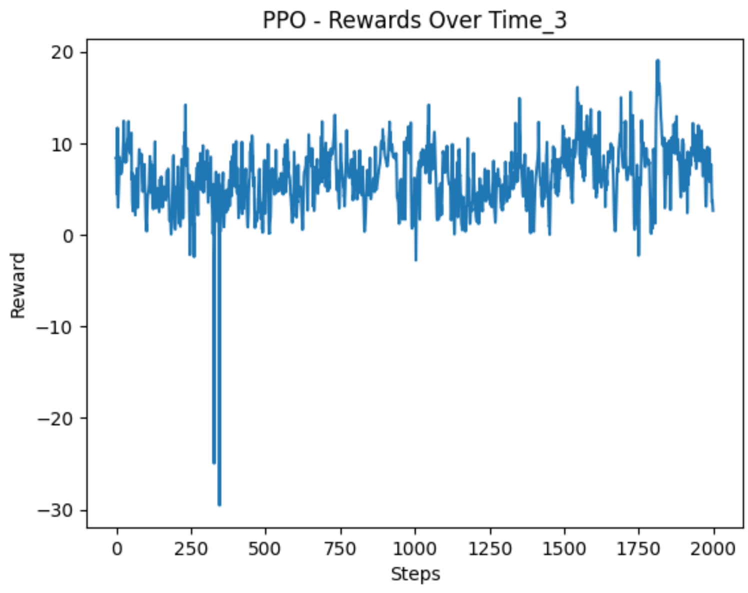

Figure 4.10: This image illustrates the reward function during training, action space is 3, the model has a good positive reward after 400 episodes.

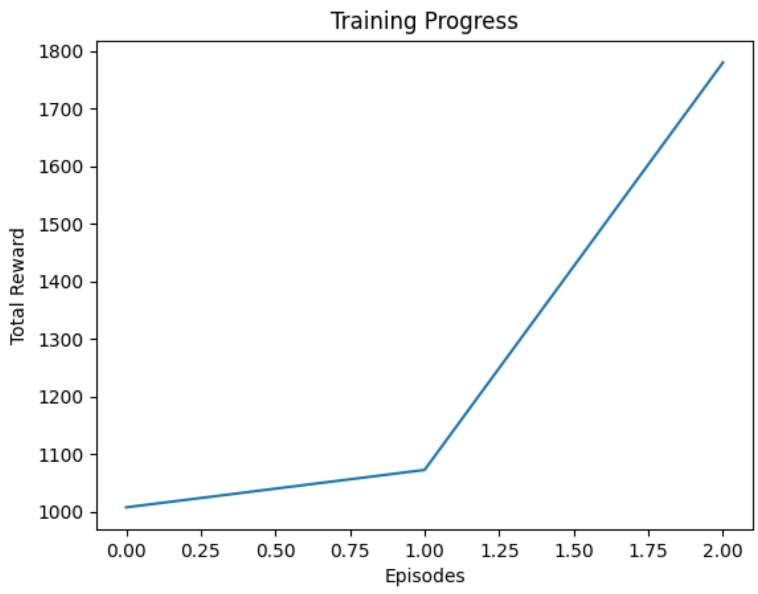

Figure 4.11: Reward of testing the agent for 2 episodes.

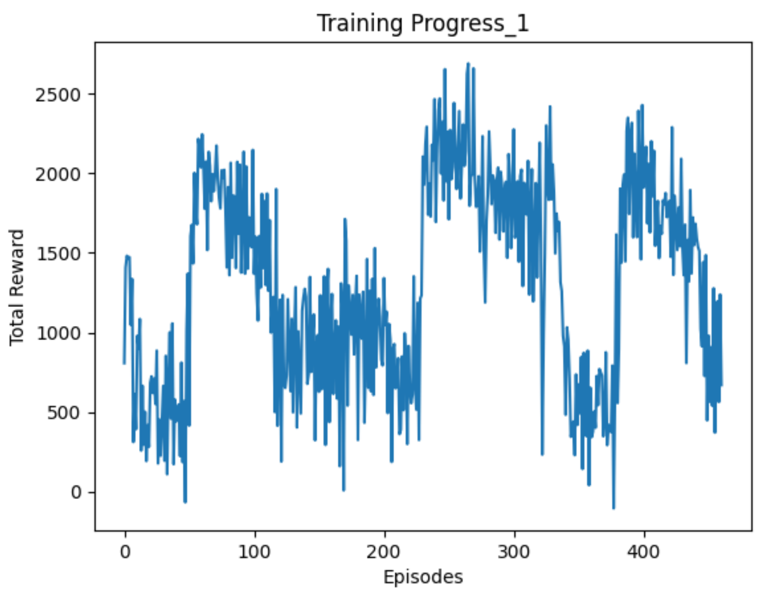

Figure 4.12: This is the reward function result during the training for 100000 time steps.

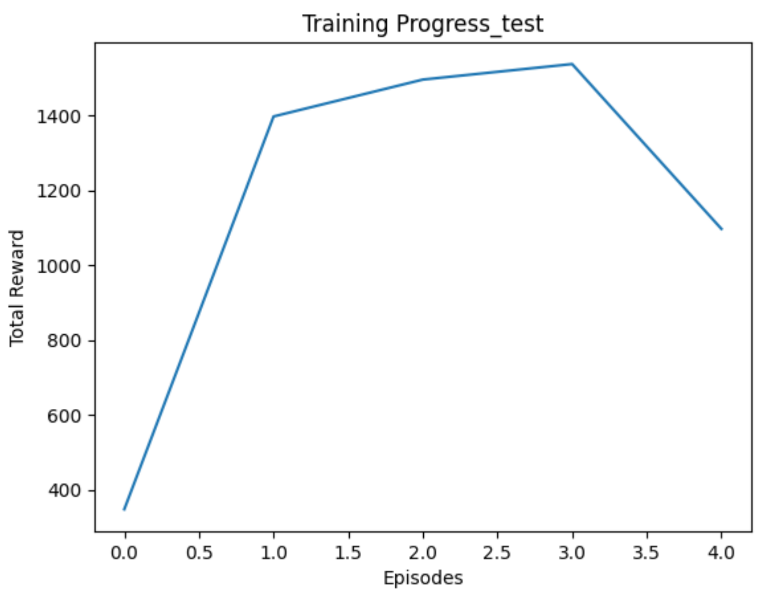

Figure 4.13: Result of testing the agent after 100k time steps training .

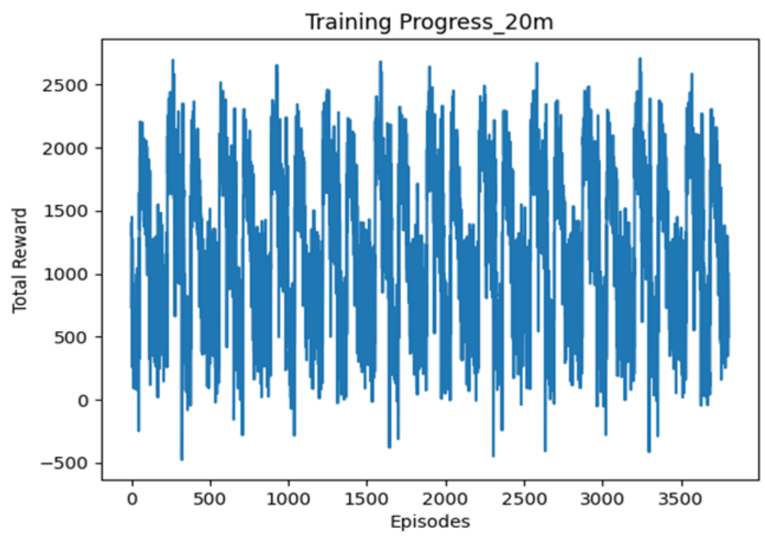

Figure 4.14: This image illustrates the reward function during the training of the agent for 1M time steps.

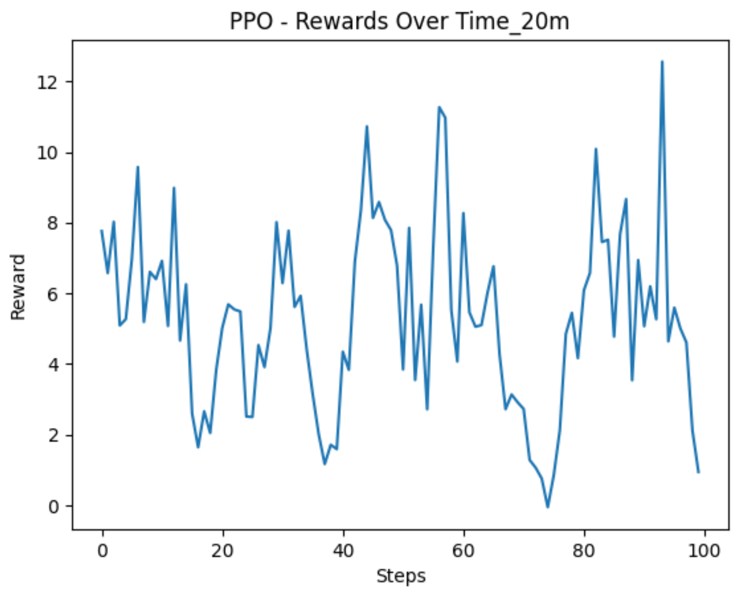

Figure 4.15: Result of testing agent after 1M time steps.

The agent was trained using the environment that was developed. The LSTM model is capable of effectively simulating the function of the exoskeleton. Subsequently, we directed the agent to learn specific tasks and enhance its performance to the greatest extent possible. Initially, a simple model and reward function were employed, but as illustrated in Figures 4.8 and 4.9, the model did not yield satisfactory results. To address this issue, the action space was modified by limiting it to three actions: remaining constant, increasing the current state by 0.1, or decreasing it by 0.1. The results of these modifications are presented in Figures 4.10 and 4.11. Moreover, a variety of hyperparameters were tested, including adjustments to the learning rate, which was modified from 3e-4 to 3e-1 to accelerate convergence. Additionally, the impact of varying batch sizes was investigated, with sizes ranging from 32, 64, and 128 to 4096, due to the extended episode lengths. A significant challenge was encountered about the episode length. As our project diverged from the conventional reinforcement learning tasks, we initially worked with a single long episode, which resulted in high computational costs and extensive processing time. To address this issue, the project was restructured with the introduction of shorter episodes, comprising 300 time steps each. Furthermore, the initial reward function was enhanced, recognizing its pivotal role in reinforcement learning, as it has the potential to markedly impact the agent's performance. The optimization of the reward function and hyperparameters necessitated a significant investment of time in trial and error. Furthermore, experiments were conducted with longer time steps, which, due to the constraints of the hardware, required several hours or even days to process. Figure 4.14 illustrates the model's reward function results in over one million time steps, a process that necessitated approximately two and a half days of training. The prolonged processing time was predominantly attributable to the limitations of computational resources and the complexity of the LSTM model, with each time step averaging 20 milliseconds, which is relatively slow.

Result interpretation

Despite the favorable outcomes, the model failed to converge. As illustrated in Figures 4.14 and 4.15, it encountered difficulties in both convergence and stabilization. While it predominantly yielded positive rewards, its performance was not consistently stable, with outcomes varying depending on the timesteps. We posit that there were multiple factors contributing to this outcome. In subsequent iterations, we sought to address these factors to enhance model stability. These factors include: Fluctuation It is postulated that the fluctuations resulting from the concatenation of disparate experiments may precipitate a precipitous shift in human torque. Furthermore, the environment is so intricate $(13200^(th)power 10) and the LSTM model predictions are so costly that$

2. Reinforcement learning is inherently challenging to stabilize, particularly due to the sensitivity of the hyperparameters and reward function to interactions and the lack of an offline dataset. This makes hyperparameter tuning time-consuming and necessitates trial and error.

3. Our simulation mode differs from the real world, and fluctuations are inevitable due to the imperfect nature of our dataset.

Future Works

Given our positive results, we plan to dedicate additional time to stabilize the agent further and achieve improved outcomes in future experiments.

Once we obtain more refined results, we intend to test our model on the actual exoskeleton to fine-tune it based on real-world applications.

We also aim to enhance the quality of our dataset to support more accurate testing in future trials.

Additionally, we plan to experiment with a wider range of deep reinforcement learning agents to compare the results better and optimize performance.

Lastly, we hope to gain access to more advanced hardware, such as GPUs, to facilitate faster training and enable more extensive trial-and-error processes, allowing us to fine-tune the parameters for stabilizing the reinforcement learning agent

4.3 Exoskeleton Test

We Plan to test our model on Exoskeleton shortly and will publish the result as well.

Chapter 5

Conclusions

In this project, we tried to make the movement of the Exoskeleton smoother by using Reinforcement Learning to cover the weakness of the PID controller in acting robotic and not humanoid. We trained our PPO agent on the Exoskeleton simulation and applied the JERK factor's minimization as a reward function for our agent. Because minimizing Jerk is the derivative of acceleration we can reach the smoother movement. As a simulation, we came up with this new idea of using Recurrent Neural Network models more specifically LSTM as our Exoskeleton simulator that could simulate the movement and Angular velocity of our Exoslsketon with excellent accuracy. Then we trained and tested our Reinforcement learning agent on top of this LSTM model as our Reinforcement learning environment and we achieved a high positive reward for our agent which means our Reinforcement learning model could learn how to produce voltage for 2 exoskeleton motors based on human movement that can provide good help for a human hand while trying to make the movement smoother.

Acronym

Bibliography

[1] https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electromyography-emg#::text=Electromyography%20(EMG)%20measures%20muscle%20response,nerve's%20stimulation%20of%20the%20muscle. [Accessed 27-08-2024].

[2] GitHub - google-deepmind/mujoco: Multi-Joint dynamics with Contact. A general purpose physics simulator. — github.com. https://github.com/google-deepmind/mujoco. [Accessed 16-08-2024].

[3] PPO &x2014; Stable Baselines3 2.4.0a8 documentation — stable-baselines3.readthedocs.io. https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html. [Accessed 30-08-2024].

[4] STT Admin. What is motion capture and which are its applications? - STT Systems — stt-systems.com. https://www.stt-systems.com/blog/what-is-motion-capture-and-which-are-its-applications/. [Accessed 27-08-2024].

[5] Kiam Heong Ang, G. Chong, and Yun Li. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005. doi:10.1109/TCST.2005.847331.

[6] Debasish Basak, Srimanta Pal, and Dipak Patranabis. Support vector regression. *Neural Information Processing – Letters and Reviews*, 11, 11 2007.

[7] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, oct 2001. doi:10.1023/A:1010933404324.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[9] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, August 2016. URL: http://dx.doi.org/10.1145/2939672.2939785, doi:10.1145/2939672.2939785.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi:10.1162/neco.1997.9.8.1735.

[11] Bernhard Jaeger and Andreas Geiger. An invitation to deep reinforcement learning, 2023. URL: https://arxiv.org/abs/2312.08365, arXiv:2312.08365.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. doi:10.1145/3065386.

[13] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing, 2018.

[14] Lucas Quesada, Dorian Verdel, Olivier Bruneau, Bastien Berret, Michel-Ange Amorim, and Nicolas Vignais. A dataset for the investigation of upper limb torque prediction from EMG signals, May 2024. doi:10.5281/zenodo.11209324.

[15] Lucas Quesada, Dorian Verdel, Olivier Bruneau, Bastien Berret, Michel-Ange Amorim, and Nicolas Vignais. Emg-to-torque models for exoskeleton assistance: a framework for the evaluation of in situ calibration. *bioRxiv*, 2024. URL: https://www.biorxiv.org/content/early/2024/06/11/2024.01.11.575155, arXiv:https://www.biorxiv.org/content/early/2024/06/11/2024.01.11.575155.full.pdf, doi:10.1101/2024.01.11.575155.

[16] Alexandra Roren, Antoine Mazarguil, Diego Vaquero-Ramos, Jean-Baptiste Deloose, Pierre-Paul Vidal, Christelle Nguyen, François Rannou, Danping Wang, Laurent Oudre, and marie-martine lefevre colau. Assessing smoothness of arm movements with jerk: A comparison of laterality, contraction mode and plane of elevation. a pilot study. *Frontiers in Bioengineering and Biotechnology*, 9, 01 2022. doi:10.3389/fbioe.2021.782740.

[17] Lowell Rose, Michael C. F. Bazzocchi, and Goldie Nejat. A model-free deep reinforcement learning approach for control of exoskeleton gait patterns. *Robotica*, 40(7):2189–2214, 2022. doi:10.1017/S0263574721001600.

[18] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL: https://api.semanticscholar.org/CorpusID:205001834.

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL: https://arxiv.org/abs/1707.06347, arXiv:1707.06347.

[20] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi:10.1109/IROS.2012.6386109.

[21] Aydin Emre Utku, Suzan Ece Ada, Muhammet Hatipoglu, Mustafa Derman, Emre Ugur, and Evren Samur. A reinforcement learning based controller to minimize forces on the crutches of a lower-limb exoskeleton, 2024. URL: https://arxiv.org/abs/2402.00135, arXiv:2402.00135.

[22] Grigorii Veviurko, Wendelin Böhmer, and Mathijs de Weerdt. To the max: Reinventing reward in reinforcement learning, 2024. URL: https://arxiv.org/abs/2402.01361, arXiv:2402.01361.